TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Ústav mechatroniky a technické informatiky

Architektura a návrh FPGA obvodů

FPGA Architecture and Design

Habilitační práce

Liberec 2010 Ing. Milan Kolář, CSc.

Abstrakt

Habilitační práce se zabývá oblastí programovatelných hradlových polí. V úvodní části jsou shrnuty charakteristické vlastnosti zakázkových integrovaných obvodů a zdůrazněny přednosti a nedostatky programovatelných obvodů. Následuje rozbor základních i speciálních architektonických prvků FPGA obvodů, stručně jsou zmíněny možnosti těchto prvků. Dále je popisována kompletní metodika návrhu FPGA od specifikace funkce systému až po implementaci obvodu. V práci je podrobně rozebrán postup systémového návrhu, kritéria posuzování jeho kvality a pravidla návrhu spolehlivých synchronních systémů. V závěru jsou diskutovány možnosti realizace aritmetických operací v FPGA obvodech, jak v pevné, tak v pohyblivé řádové čárce.

Klíčová slova: hradlové pole, architektura FPGA, metodika návrhu, systémový návrh, HDL, implementace aritmetických operací

Abstract

The inaugural dissertation deals with the sphere of programmable gate arrays. In the preliminary part characteristic features of custom integrated circuits are summarized and preferences and disadvantages of programmable circuits are underlined. FPGA basic and special architectural elements analysis follows, there are presented options of these elements in brief. Subsequently there is a complex methodology of FPGA design described from system function specification up to the circuit implementation. In the dissertation system design procedure, criterions of this quality assessing and rules of the reliable synchronous systems design are analysed. Finally, variants of the arithmetic operations realization on FPGA circuits are discussed, both fixed and floating point.

Keywords: gate array, FPGA architecture, design methodology, system design, HDL, implementation of arithmetic operations

Obsah

Seznam zkratek

AES	*Advanced Encryption Standard* – druh šifrovacího standardu
ALM	*Adaptive Logic Module* – adaptivní logický modul
ALU	*Arithmetic Logic Unit* – aritmeticko-logická jednotka
ALUT	*Adaptive LUT* – adaptivní vyhledávací tabulka
ASIC	*Application Specific Integrated Circuit* – aplikačně specifický IO
BGA	*Ball-Grid Array* – druh pouzdra integrovaného obvodu
BLVDS	*Bus LVDS* – druh vstupně-výstupní sběrnice
CE	*Clock Enable* – odblokování hodinového signálu
CLB	*Configurable Logic Block* – konfigurovatelný logický blok
CMOS	*Complementary Metal Oxide Semiconductor* – druh vstupně-výstupního standardu, druh výrobní technologie
CMT	*Clock Management Tile* – druh hodinového manažeru
CP	*Change Pump* - nábojová pumpa
CPI	*Cycles Per Instruction* – počet taktů na instrukci
CPLD	*Complex PLD* – komplexní obvody PLD
CPU	*Central Processor Unit* – centrální procesorová jednotka
DC	*Direct Current* – stejnosměrný proud
DCI	*Digitally Controlled Impedance* – nastavení impedančního přizpůsobení
DCM	*Digital Clock Manager* – druh hodinového manažeru
DDR	*Double Data Rate* – druh dynamické paměti RAM
DFS	*Digital Frequency Synthesis* – digitální frekvenční syntéza
DLL	*Delay Locked Loop* – druh smyčky fázového závěsu
DMIPS	*Dhrystone MIPS* – jednotka výkonnosti
DSP	*Digital Signal Processing* – zpracování digitálního signálu
EDIF	*Electronic Design Interface Format* – druh standardizovaného formátu pro výměnu dat mezi návrhovými systémy
EDK	*Embedded Development Kit* – vývojové prostředí firmy Xilinx
EEPROM	*Electrically Erasable PROM* – elektricky mazatelná paměť PROM
ESR	*Equivalent Series Resistance* – reálná složka impedance
FA	*Full Adder* – úplná sčítačka
FBGA	*FineLine BGA* – druh pouzdra integrovaného obvodu
FFT	*Fast Fourier Transform* – rychlá Fourierova transformace
FIFO	*First In First Out* – druh neadresovatelné paměti
FP	*Floating Point* – pohyblivá řádová čárka
FPGA	*Field Programmable Gate Array* – programovatelné hradlové pole
FPSLIC	*Field Programmable System Level Integrated Circuit* – série obvodů firmy Atmel

FX	*Fixed Point* – pevná řádová čárka
GAL	*Generic Array Logic* – druh programovatelného obvodu
HDL	*Hardware Description Language* – jazyk pro návrh technických prostředků
HSTL	*High Speed Transceiver Logic* – druh vstupně-výstupního standardu
HW	*Hardware* – technické prostředky
IEEE	*Institute of Electrical and Electronics Engineers* – normotvorná organizace
I2C	*Inter-Integrated Circuit* – druh sériové sběrnice
IC	*Instruction Count* – počet instrukcí
ICAP	*Internal Configuration Access Port* – druh konfiguračního rozhraní
IO	Integrovaný obvod
I/O	*Input/Output* – vstupně-výstupní
IP	*Intellectual Property* – duševní vlastnictví
ISA	*Instruction Set Architecture* – architektura souboru instrukcí
ISE	*Integrated Synthesis Environment* – vývojové prostředí firmy Xilinx
ISF	*In System Flash* – systém obsahující flash paměť
ISP	*In System Programming* – programovatelný v systému
JTAG	*Joint Test Action Group* – druh standardizovaného rozhraní
LAB	*Logic Array Block* – blok logických polí
LC	*Logic Cell* – logická buňka
LE	*Logic Element* – logický element
LF	*Loop Filter* - smyčka filtru
LIFO	*Last In First Out* – druh neadresovatelné paměti
LPM	*Library of Parameterized Modules* – knihovna parametrizovatelných modulů
LSB	*Least Significant Bit* – nejméně významný bit
LVCMOS	*Low Voltage CMOS* – druh vstupně-výstupního standardu
LVDS	*Low Voltage Differential Signaling* – druh vstupně-výstupního standardu
LVPECL	*Low Voltage Positive Emitter Coupled Logic* – druh vstupně-výstupního standardu
LVTTL	*Low Voltage Transistor Transistor Logic* – druh vstupně-výstupního standardu
LUT	*Look Up Table* – vyhledávací tabulka
MAC	*Multiply and Accumulates* – jednotka pro násobení a mezisoučty
MIPS	*Million Instructions Per Second* – milióny instrukcí za sekundu
MLAB	*Memory Logic Array Block* – druh logických bloků
MMAC	*Million Multiply Accumulates* – milióny násobení a mezisoučtů
MMCM	*Mixed-Mode Clock Manager* – druh hodinového manažeru
MMU	*Memory Management Unit* – jednotka správy paměti
MNOS	*Metal Nitride Oxide Semiconductor* – druh unipolárního tranzistoru
MPGA	*Mask Programmable Gate Array* – hradlová pole programovatelná maskou
MSB	*Most Significant Bit* – nejvýznamnější bit

MSPS	*Million Samples Per Second* – milión vzorků za sekundu
MTBF	*Mean Time Between Failures* – střední doba bezporuchového provozu
NaN	*Not a Number* – nečíselná hodnota
OCT	*On-Chip Termination* – blok pro nastavení impedančního přizpůsobení
OE	*Output Enable* – signál pro režim vysoké impedance
OTP	*One Time Programmable* – jednou programovatelná (paměť)
PAL	*Programmable Array Logic* – programovatelné logické pole
PCI	*Peripheral Component Interconnect* – sběrnice pro připojení periferií
PFD	*Phase-Frequency Detector* – fázově-frekvenční detektor
PIP	*Programmable Interconnection Point* – programovatelné připojovací body
PLA	*Programmable Logic Array* – programovatelné logické pole
PLB	*Processor Local Bus* – procesorová sběrnice uvnitř FPGA
PLCC	*Plastic Leaded Chip Carrier* – druh pouzdra integrovaného obvodu
PLD	*Programmable Logic Device* – programovatelná logická součástka
PLICE	*Programmable Low Impedance Circuit Element* – druh antipojistek
PLL	*Phase Locked Loop* – smyčka fázového závěsu
POR	*Power-On Reset* – blok pro zajištění nulování při náběhu napájení
PROM	*Programmable Read Only Memory* – jednorázově programovatelná paměť
PSM	*Programmable Switch Matrix* – programovatelná přepínací matice
RAM	*Random Access Memory* – paměť s libovolným přístupem
RC	*Resistor Capacitor* – prvek obsahující rezistor a kapacitor
RISC	*Reduced Instruction Set Computer* – počítač s redukovanou instrukční sadou
RTL	*Register Transfer Level* – úroveň meziregistrových přenosů
SHA	*Secure Hash Algorithm* – šifrovací algoritmus
SMIP	*Switching Matrix Interconnection Point* – propojovací body matice přepínačů
SOPC	*System On a Programmable Chip* – vývojové prostředí firmy Altera
SPI	*Serial Peripheral Interface* – druh sériového rozhraní
SRAM	*Static RAM* – statická paměť RAM
SSTL	*Stub Series Terminated Logic* – druh vstupně-výstupního standardu
STA	*Static Timing Analysis* – statická časová analýza
SW	*Software* – programové prostředky
TDM	*Time Division Multiplex* – časový multiplex
TQFP	*Thin Quad Flat Pack* – druh pouzdra integrovaného obvodu
VCO	*Voltage Controlled Oscillator* – napětím řízený oscilátor
VHDL	*VHSIC Hardware Description Language* – druh jazyka HDL
VHDL-AMS	*VHDL Analog & Mixed Signals* – VHDL pro analogové a smíšené obvody
VHSIC	*Very High Speed Integrated Circuit* – velmi rychlý integrovaný obvod
VLSI	*Very Large Scale Integration* – velmi velká integrace
WE	*Write Enable* – povolení zápisu

Seznam symbolů

$A(B)$	transformace binárního čísla B (aditivní kód)
a, b	obecné operandy aritmetických operací
B	obecné binární číslo
c	obecný výsledek aritmetických operací
c_d	hodnota dělicího poměru děličky Cn
C	parazitní kapacita [F]
C_1, C_2	konstanty závislé na typu obvodu a na použité výrobní technologii [s]
CPI	průměrný počet taktů na instrukci
d	kódová vzdálenost
$D(B)$	transformace binárního čísla B (doplňkový kód)
E	exponent FP čísla
exp	hodnota exponentu v aditivním kódu
f	pracovní kmitočet [Hz]
f_{clk}	kmitočet hodinového signálu v časové doméně [Hz]
f_{data}	kmitočet změny dat vstupního asynchronního signálu [Hz]
f_{in}	kmitočet vstupního signálu [Hz]
f_{pll}	kmitočet na výstupu PLL [Hz]
f_{vco}	kmitočet na výstupu VCO [Hz]
i	obecný výčtový index
I	napájecí proud [A]
I_{out}	výstupní proud regulátoru [A]
$I(B)$	transformace binárního čísla B (inverzní kód)
IC	počet instrukcí
k	minimální počet bitů stavového slova
k_{out}	počet vzájemně posunutých signálů na výstupu VCO
K	konstanta posunutí u aditivního kódu
m	počet bitů zlomkové části řádové mřížky
m_d	hodnota dělicího poměru děličky ve zpětné vazbě smyčky závěsu
M	modul řádové mřížky
M_{IEEE}	absolutní hodnota mantisy bez nejvyšší jedničky
M_N	mantisa FP čísla
$MTBF$	střední doba bezporuchového provozu daného synchronizátoru [s]
$MTBF_C$	střední doba bezporuchového provozu celého návrhu [s]
n	počet bitů celé části řádové mřížky
n_d	hodnota dělicího poměru předděličky na vstupu fázového závěsu
n_s	počet vnitřních stavů stavového automatu
N	délka řádové mřížky (velikost datového slova)

N_C	počet všech synchronizátorů v návrhu
N_T	počet taktů na zpracování jedné úlohy
P	výkonnost výpočetního systému [s^{-1}]
P_D	dynamický příkon [W]
P_S	statický příkon [W]
P_Z	ztrátový výkon [W]
$P(B)$	transformace binárního čísla B (přímý kód se znaménkem)
Q_C	poruchovost celého návrhu [s^{-1}]
r	obecné celé číslo
s_i	číslice i-tého řádu binárního čísla S_{FX}
S	znaménkový bit
S_{FX}	obecné binární číslo v doplňkovém kódu (signed)
S_M	měřítko čísla v pevné řádové čárce
t_h	doba přesahu [s]
t_{met}	doba přechodu z metastabilního stavu do definované logické hodnoty [s]
t_{rr}	doba zotavení po resetu [s]
t_s	doba předstihu [s]
T_{clk}	perioda hodinového kmitočtu [s]
T_{CPU}	čas zpracování jedné úlohy v procesoru [s]
T_u	čas zpracování dané úlohy [s]
T_{vco}	perioda signálu na výstupu VCO [s]
U	napájecí napětí [V]
U_{in}	vstupní napětí regulátoru [V]
U_{out}	výstupní napětí regulátoru [V]
x_i	číslice i-tého řádu binárního čísla X_{FX}
X_{FX}	obecné nezáporné binární číslo (unsigned)
X_{FP}	obecné číslo ve formátu pohyblivé řádové čárky
ε	jednotka řádové mřížky
φ_f	fázový posuv s jemným rozlišením na výstupu VCO [s]
φ_c	fázový posuv s hrubým rozlišením na výstupu VCO [s]

Seznam obrázků

Seznam tabulek

Úvod

Složitost integrovaných obvodů (IO), jak analogových, tak číslicových, dlouhodobě výrazně roste. Již zhruba 40 let platí tzv. Moorův zákon, podle kterého se počet tranzistorů na čipu exponenciálně zvyšuje – zdvojnásobí se cca každých 18 měsíců. Zpočátku se zdálo, že návrh IO bude záležitostí poměrně úzkého okruhu návrhářů soustředěných přímo v technologických firmách, které IO vyrábějí. Se vzrůstající velikostí čipů měli „technologičtí“ návrháři stále větší problém veškeré tranzistory na čipu smysluplně využít potřebnou logikou. Vzhledem k vysoké ceně vývoje bylo nutné na čip integrovat dostatečně univerzální funkce, a tím zajistit prodej velkých sérií. Tento požadavek není problémem např. u běžných procesorů nebo pamětí, ale v oblasti logických obvodů je obtížné nalézt rozsáhlé univerzální funkce. Proto bylo třeba vytvořit podmínky pro částečný či úplný přesun návrhu IO k jejich uživatelům (výrobcům elektroniky), kteří nejlépe vědí, jaké funkce od čipů očekávají.

Prudký rozvoj jak mikroelektronických technologií, tak rozvoj výpočetní techniky vnesl do metodiky návrhu IO zásadní obrat. V podstatě došlo k přesunu systémového návrhu přímo k uživatelům IO, vznikají zakázkové IO. Snahou bylo snižovat náklady na vývoj a ekonomičnosti dosáhnout čipem přesně přizpůsobeným požadované funkci, kterou definují přímo budoucí uživatelé (systémoví návrháři). Zmíněný vývoj technologií umožňuje vytvářet nejen rozsáhlejší, ale také rychlejší integrované obvody. Zrychlování funkcí sebou přináší nejen nové aplikační možnosti, ale také nové problémy při jejich návrhu. Na systémové návrháře jsou kladeny stále vyšší požadavky z hlediska jejich znalostí a zkušeností, aby navrhli výsledný spolehlivě fungující systém. Proto si návrháři musí osvojovat stále nové techniky, zásady, metodiku návrhu či moderní architektury obvodů.

Největší rozvoj z kategorie číslicových IO v současnosti zažívají programovatelná hradlová pole (FPGA), na jejichž návrh se v této práci zaměříme. Díky jejich vzrůstající velikosti a rychlosti a současně klesající ceně se aplikační možnosti FPGA obvodů přesouvají od časově kritických úloh ke komplexnímu zpracování číslicových signálů a přebírají úkoly dříve vyhrazené levnějším, ale pomalejším, procesorům.

Z výše uvedených důvodů je třeba, aby systémoví návrháři různých technických oborů dobře znali moderní prvky architektur programovatelných obvodů, jejich přednosti i omezení. Současně musí ovládat metodiku a pravidla návrhu spolehlivých synchronních systémů a musí být schopni optimálně nastavit parametry nástrojů syntézy v návrhových systémech. K tomu by měla přispět i tato práce.

1 Rozdělení obvodů

V úvodu jsme se zmínili o rychlém rozvoji oblasti architektur a návrhu číslicových obvodů. Vlivem tohoto rozvoje vznikají nové odborné termíny, které se však nepoužívají v různých literaturách nejednotně (nejsou ustálené). Proto si nejen pro účely této práce utřiďme některé názvy, zkratky a kategorie. Začněme členěním **číslicových IO**, které podle [1] rozdělujeme na:

- ***standardní logické obvody*** – obvody nízké a střední integrace (například řada 7400 v technologii TTL nebo 4000 v technologii CMOS),
- ***univerzální procesory*** – obvody řízené instrukcemi,
- ***zakázkové obvody*** – obvody přesně navržené podle požadavků zákazníka (uživatele).

Kategorie **zakázkových obvodů** je velice rozmanitá a některé její druhy je obtížné jednoznačně zařadit. Nejčastěji se tyto obvody člení na:

- ***ASIC obvody*** (Application Specific Integrated Circuits) – u těchto druhů obvodů je nutné podle přání uživatele navrhnout veškeré masky technologického procesu. Tyto obvody se také označují jako *zakázkové* (Custom) obvody v užším smyslu slova a jejich návrh bývá omezen nějakou knihovnou – např. *standardní buňky* (Standard Cells). Do skupiny ASIC obvodů řadíme i plně zakázkové (Full Custom) obvody. V některých literaturách se pojmem ASIC označují veškeré zakázkové obvody, ale v tomto významu jej používat nebudeme.
- ***Polozakázkové IO*** (Semi-custom Integrated Circuits) – charakteristickým znakem těchto obvodů je univerzální „polotovar“, připravený výrobcem technologie, na kterém jsou připraveny samostatné elektronické prvky různé složitosti (tranzistory, logická hradla, registry). Podle požadavků koncového uživatele se pak navrhnou pouze všechny propojovací masky. Někdy se tyto obvody označují jako *hradlová pole* (Gate Arrays). Do této kategorie je možné zařadit i nové obvody označované jako *Structured ASIC* – na čipu jsou připraveny funkční bloky (logika, registry, paměti, procesory, řadiče rozhraní či jiná komplexnější jádra), které se podle požadavků zákazníka perzonifikují jednou nebo dvěma metalickými maskami. Někdy se tyto obvody označují MPGA (Mask Programmable Gate Array) – hradlová pole programovatelná maskou.
- ***Programovatelné IO*** (Programmable Integrated Circuits) – do této kategorie řadíme obvody, které lze programovat přímo uživatelem. Nemáme však na mysli součástky ovládané programem prostřednictvím instrukcí, ale „nastavování“ propojek mezi jednotlivými vnitřními vodiči (buď jednorázově natrvalo nebo s možností přeprogramování). V každém případě výroba těchto obvodů u výrobců technologie není závislá na budoucím použití.

Výběr druhu příslušného zakázkového obvodu závisí především na předpokládané výrobní sérii. Je zřejmé, že cena vývoje plně zakázkového ASIC obvodu bude mnohem vyšší než u programovatelných obvodů a je nutné ji „rozpustit“ do počtu vyrobených kusů. Vlastní náklady na jeden vyrobený kus však pak budou oproti programovatelným součástkám velmi nízké. Díky dokonalému využití možností polovodičové technologie dosáhneme u ASIC obvodů menší plochy čipu, a tím i lepších funkčních parametrů. Jakákoli budoucí změna funkce obvodu je ale bez nového návrhu veškerých masek nemožná.

Kategorie polozakázkových obvodů je díky výraznému rozvoji programovatelných obvodů spíše na ústupu. Ujímají se pouze zmiňované Structured ASIC, které se vnitřní strukturou podobají FPGA obvodům, ale propojovací sítě nejsou programovatelné. Nejvýznamnější výrobci programovatelných obvodů nabízejí následující možnost - uživatel si navrhne a odzkouší návrh v běžném FPGA obvodu a pak jej předá výrobci, který mu cca do 3 měsíců dodá architektonicky shodné ekvivalenty, které již nejsou reprogramovatelné, ale na druhou stranu jsou velice levné (z hlediska ceny za kus). Xilinx nazývá tyto obvody *EasyPath* [2], Altera pak *HardCopy* [3]. Tím se v podstatě eliminuje určitá nevýhoda zakázkových a polozakázkových obvodů – systémový návrhář obvodů již není součástí výrobního procesu IO, a proto nemusí mít specializované technologické znalosti.

Vzhledem k neustálému vývoji technologií (zvětšování hustoty integrace), a s tím související zvětšování množství funkcí v IO, se stávají programovatelné obvody stále používanějšími. Výhodou programovatelných součástek je zejména jejich snadná dostupnost, nízké počáteční náklady, relativně jednoduchý návrh pomocí snadno dostupných návrhových systémů a snadná změna návrhu. Další příčinou jejich obliby je i velký tlak na zrychlování vývoje a uvádění nových inovovaných výrobků na trh. Jen v oblasti programovatelných obvodů se vyvinula celá řada architektur, které jsou základním kritériem jejich dělení. Zhruba se **programovatelné logické obvody** dělí na obvody:

- ***PLD*** (Programmable Logic Device) – základem je dvoustupňová architektura polí AND a OR, vhodná pro realizaci kombinačních logických funkcí. Doplněním této architektury o klopné obvody umožňují PLD realizovat jednodušší synchronní sekvenční obvody. Podle toho, které z polí je programovatelné, dále tyto obvody rozdělujeme na:
  - **PROM** (Programmable Read Only Memory) – součinové pole je pevné, součtové pole je programovatelné. Jejich nevýhodou je relativně malá hustota realizované logiky vzhledem k velikosti programovatelné matice.
  - **PAL** (Programmable Array Logic) – programovatelné je součinové pole, součtové pole je pevné. Tento typ PLD obvodů byl ve své době velmi úspěšný, zvláště jeho varianta doplněná na výstupu o makrobuňky, označovaná jako GAL (Generic Array Logic).

- **PLA** (Programmable Logic Array) – obě pole (součinové i součtové) jsou programovatelná. Díky dvěma programovatelným maticím v kaskádě jsou relativně drahé a pomalé.

- ***CPLD*** (Complex PLD) – složitější architektury vycházející z obvodů PLD. S rostoucí složitostí logických funkcí není možné neustále zvětšovat programovatelné matice PLD obvodů, a proto CPLD vznikají skládáním více PLD (nejčastěji GAL) obvodů do maticových či vrstevnatých struktur propojených centrální propojovací maticí.
- ***FPGA*** (Field Programmable Gate Array) – základem je pravidelná struktura programovatelných logických bloků doplněná sítí propojovacích vodičů a spínacích matic, podrobnější popis architektury je v kap. 2.

Obecně jsou programovatelné logické obvody pro běžné vývojáře elektronických systémů ze všech zakázkových obvodů nejdostupnější, a tedy také nejpoužívanější. V praxi se v současné době používají převážně jen obvody CPLD (pro jednodušší a středně složité funkce) a FPGA (pro složitější logické systémy). Díky rostoucí složitosti systémů budou stále více převažovat obvody FPGA.

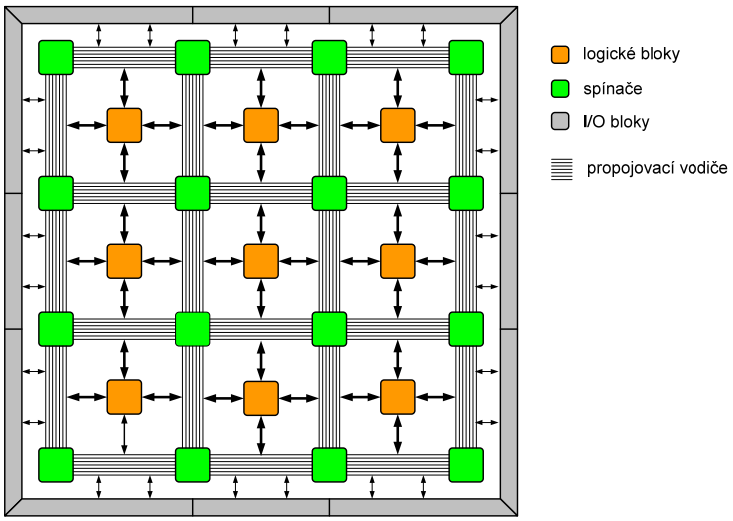
Na trhu s programovatelnými obvody nepůsobí příliš velké množství výrobců a významný podíl má pouze několik z nich. Podle [4] je dominantní firma Xilinx s 50procentním podílem, druhá je Altera s 36 %, třetí Lattice Semiconductor, čtvrtý Actel (oba zhruba se 6 %). Zbývající 2 % mají ostatní výrobci, mezi kterými převažuje QuickLogic. Z tohoto důvodu se budeme nejvíce věnovat právě produktům prvních dvou firem a rovněž budeme uvádět jejich používanou terminologii (která je bohužel mnohdy rozdílná).

2 Architektura FPGA obvodů

Obvody FPGA jsou v dnešní době ze všech programovatelných obvodů nejpoužívanější a jsou vhodné zejména v aplikacích vyráběných v menších sériích (řekněme řádově pod 10 000 kusů). Programují se přímo u zákazníka (field – „v poli“), nikoliv při výrobě. Cenou za tuto výhodu je složitější struktura čipu. Abychom se v pozdějších kapitolách mohli zabývat specifiky návrhů programovatelných hradlových polí, je třeba nejprve důkladněji rozebrat jejich architekturu. Architektura FPGA je určena zejména strukturou logických bloků, jejich rozmístěním, topologií propojovacích vodičů a výběrem technologie programování. Proto se nyní věnujme jednotlivým strukturním prvkům architektury, a to jak standardním, tak i speciálním, které nemusí být součástí každého FPGA obvodu.

2.1 Základní prvky architektury

Mezi základní prvky architektury patří programovatelné logické bloky, propojovací sítě a vstupně-výstupní (I/O) bloky. FPGA jsou často organizovány ve formě *dvourozměrných matic* logických bloků s propojovacími vodiči probíhajícími v kanálech mezi logickými bloky (obr. 1) nebo jako *řádková architektura* s logickými bloky v řádcích a propojovacími vodiči probíhajícími v kanálech mezi řádky (podobně jako u technologie standardních buněk používané při realizaci ASIC). Z hlediska architektury je možná i *hierarchická architektura*



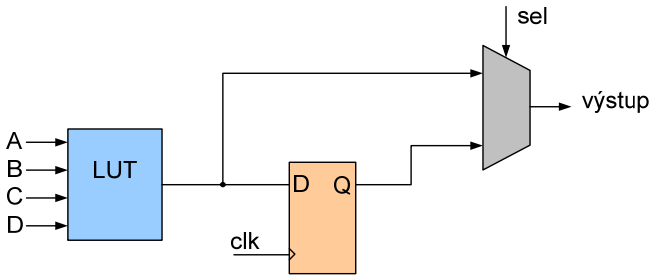
Obr. 1: Základní architektura FPGA obvodů

(používaná již u některých obvodů CPLD), kde několik logických bloků tvoří společně s lokální propojovací strukturou jedno pole logických bloků, které potom používá stejné propojovací vodiče k připojení na obecnou propojovací strukturu.

2.1.1 Programovatelné logické bloky

Logické bloky, označované nejčastěji **CLB** (Configurable Logic Block – Xilinx) nebo **LAB** (Logic Array Block – Altera), slouží k vytváření logických, aritmetických a paměťových funkcí v FPGA obvodech. Jsou složeny z *logických buněk*, označovaných **LC** (Logic Cell – Xilinx) nebo **LE** (Logic Element – Altera). Tyto buňky jsou natolik standardní, že se staly základem i pro porovnávání velikosti FPGA obvodů. Dřívější srovnávání počtu tranzistorů na čipu nebo počet systémových dvouvstupových ekvivalentních hradel je (u struktur určených převážně pro realizaci synchronních sekvenčních systémů) mnohdy zavádějící.

Logické buňky jsou základním prvkem k realizaci kombinační i sekvenční logiky. Jejich struktura se neustále vyvíjí a není jednoduché ji nějak zobecnit. Skládají se v principu z obvodu realizujícího jednoduchou, typicky čtyřvstupovou, kombinační funkci (označuje se jako **LUT** – Look Up Table, vyhledávací tabulka), dále z klopného obvodu (nejčastěji jednoho) pro realizaci synchronní sekvenční části a nakonec z lokálního propojení (obr. 2).



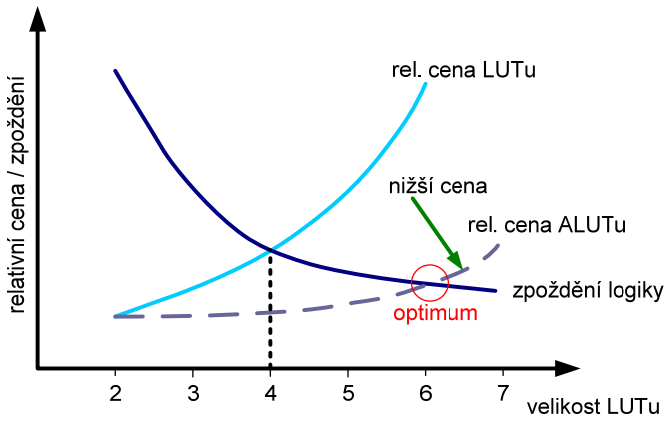
Obr. 2: Základní logická buňka

Výstupní logická výhybka umožňuje volbu kombinačního nebo registrového výstupu. Čtyřvstupový LUT je schopen realizovat všechny kombinační logické funkce čtyř proměnných. Obecně je funkce LUTů při realizaci logické funkce založena na dvou podobných principech – jednak na principu SRAM (funkce je uložená v paměti a vstupy představují adresu konkrétní paměťové buňky) a jednak na principu multiplexoru (vstupy jsou připojeny na jeho adresovací vstupy, na datových vstupech multiplexoru je v registrech uložena požadovaná logické funkce). V obou případech je třeba pro realizaci *r*-vstupového LUTu potřeba 2^r paměťových členů (*r* je obecné přirozené číslo). S každým zvětšením „*r*“ o jedničku se velikost paměťové matice zdvojnásobí. Klopné obvody v logických elementech jsou většinou typu D s hranovým řízením (*D flip-flop*), někdy umožňují i hladinové řízení (*D latch*), mají vstup CE (clock enable) a minimálně jeden další konfigurovatelný vstup jako nastavovací-nulovací v synchronním nebo asynchronním režimu. Součástí logických buněk mohou být např. i podpůrné struktury (multiplexory či logická hradla) pro efektivní realizaci aritmetických operací (sčítačka, řetězec rychlého přenosu aj.).

Výše popsaná logická buňka však není pro všechny navrhované logické systémy nejvhodnější, a proto existuje i množství jejích variant. Jistě si mnohý návrhář položí otázku,

jaká je optimální velikost této buňky. Velikost logické buňky určuje tzv. **granularitu** programovatelného obvodu. Obecně lze ale říci, že logické buňky v FPGA obvodech jsou jednodušší ve srovnání s např. makrobuňkami typickými pro obvody PLD či CPLD. Obvody s makrobuňkami (označují se také jako obvody s *hrubozrnnou strukturou*) byly především určené pro jednoprůchodovou realizaci logických funkcí. Jednoprůchodovou realizací rozumějme realizaci funkce jedinou makrobuňkou bez nutnosti propojovat větší počet makrobuněk pro realizaci této funkce. Samozřejmě se ale víceprůchodové zapojení nevylučuje a je možné. Naopak u obvodů FPGA se pro realizaci logické funkce počítá s propojením většího počtu jednoduchých buněk (mluvíme tedy o obvodech s *jemnozrnnou strukturou*). Výhodou jednodušších buněk je obecně snadnější syntéza, menší a lépe odhadnutelné zpoždění, lepší využitelnost logiky, a s tím související menší spotřeba energie celého logického obvodu. Proto tyto struktury v současných FPGA obvodech výrazně převažují.

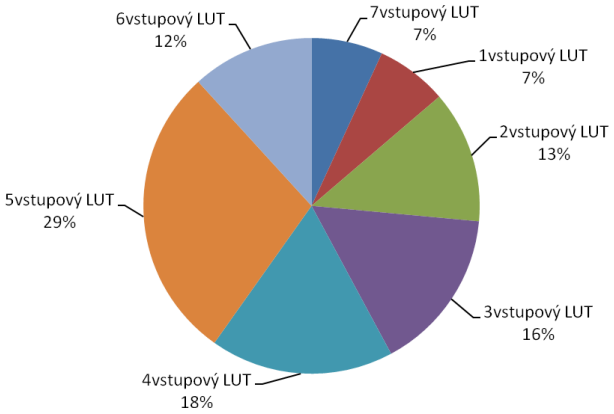
Jak již bylo naznačeno, typická logická buňka obsahuje 4vstupový LUT. Pokud potřebujeme zrealizovat např. jen dvouvstupovou kombinační funkci, zůstane zbytek LUTu nevyužitý. Naopak při potřebě zrealizovat např. 6vstupovou funkci, je třeba do kaskády zapojit dva LUTy, čímž se zvyšuje doba průchodu signálu ze vstupu na výstup, a zhoršují se dynamické vlastnosti. Čtyřvstupový LUT se tedy může zdát jakýmsi kompromisem mezi efektivitou využití logiky (cenou) a velikostí zpoždění logiky (viz obr. 3).



Obr. 3: Závislost relativní ceny a zpoždění na velikosti LUTu

Na základě statistické analýzy (vycházející z více jak 150 tisíc experimentů) [5] jsou třeba v logických systémech nejvíce 5vstupové LUTy (29 %), 18 % 4vstupových a 16 % 3vstupových LUTů (viz obr. 4). Z tohoto přehledu se jeví, že by bylo výhodnější používat raději 5vstupové LUTy, protože se jich používá procentuálně nejvíce. Více jak polovina všech potřebných LUTů je ale méně než 5vstupová, a proto by ve většině LUTů zůstalo velké množství nevyužité logiky. Proto raději případné pěti a vícevstupové LUTy složíme ze dvou 4vstupových.

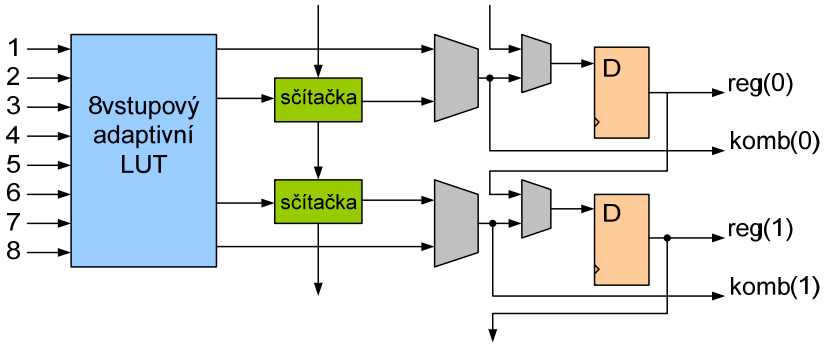
V dnešních nejmodernějších FPGA obvodech je trendem spíše návrat k větším logickým buňkám (k hrubozrnnějším strukturám), však větší variabilita těchto buněk zaručuje jejich efektivnější využití z hlediska logiky. Zejména adaptivnost jejich LUTů umožňuje realizovat „méněprůchodové“ logické funkce, a tím zlepšovat dynamické vlastnosti celého systému.



Obr. 4: Potřeba *r*-vstupových LUTů v logických systémech

Nové struktury dovolují využívat paměťové kapacity LUTů k realizaci distribuované paměti a výstupní klopné obvody k realizaci posuvných registrů. Obecně je velké množství registrů typické pro moderní FPGA architektury (jsou to tzv. *register-rich architectures*). Detailní popis možností těchto logických buněk překračuje cíle této práce, přesto stručně naznačíme jejich možnosti.

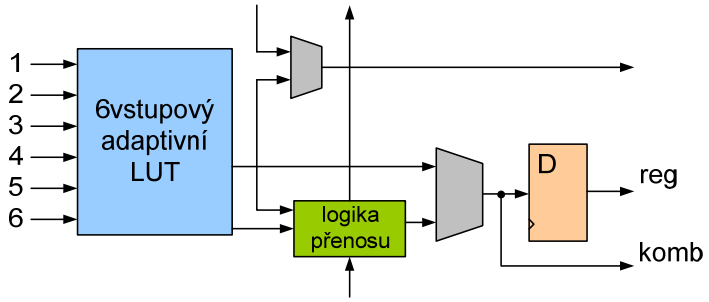
Zhruba před pěti lety přišla firma Altera u obvodů FPGA řady Stratix II s novou strukturou logických buněk nazvanou **ALM** (Adaptive Logic Module) [6]. Jedná se v podstatě o konfigurovatelný logický modul skládající z 8vstupového fragmentovatelného LUTu se 64bitovou pamětí, dvou dedikovaných sčítaček, dvou hranově řízených registrů a doplňkové logiky (obr. 5). Do tohoto modulu lze implementovat vybrané 7vstupové funkce, všechny 6vstupové funkce, dvě nezávislé funkce s menším počtem vstupů (4+4 a 5+3 vstupů) nebo dvě částečně závislé funkce (5+4, 5+5, 6+6 vstupů). Vložené sčítačky slouží v aritmetickém módu k efektivní realizaci vícebitových sčítaček, čítačů nebo komparátorů. Deset modulů ALM potom tvoří LAB, resp. nově se tyto bloky označují MLAB (Memory



Obr. 5: Adaptivní logický modul firmy Altera

Logic Array Block), což souvisí s využíváním těchto bloků jako distribuované paměti (viz kapitola 2.2.1.).

Firma Xilinx nastolila podobný trend a do svých FPGA typu Virtex-5 implementuje podobné bloky nazývané **LUT-FlipFlop Pair** [7]. Blok obsahuje 6vstupový LUT se 64bitovou pamětí, carry logiku, multiplexor a klopný obvod (obr. 6). LUT je možné také



Obr. 6: LUT-FlipFlop Pair firmy Xilinx

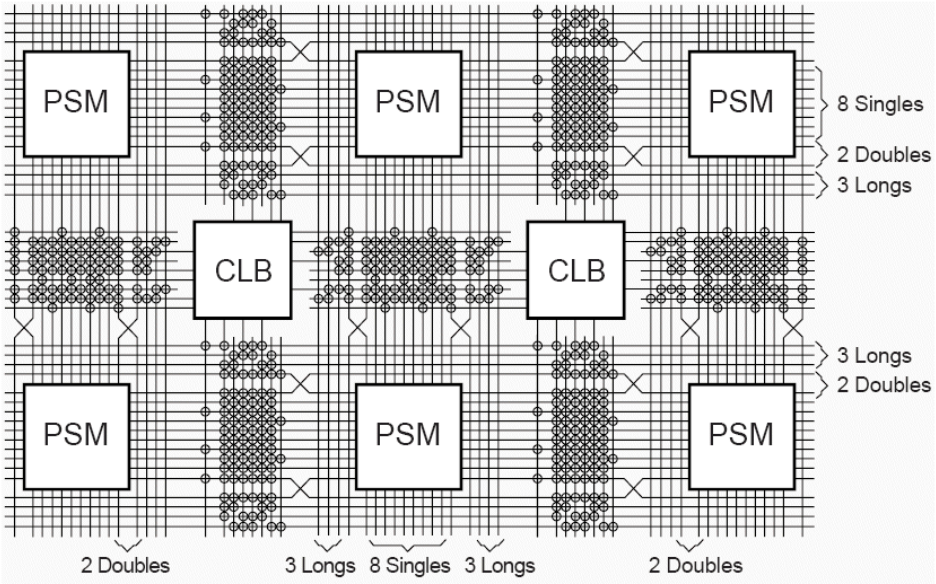
rozdělit na dvě samostatné části, ale vlivem menšího počtu vstupů má omezenější možnosti ve srovnání s ALM. Například při současné realizaci pětivstupového a třívstupového LUTu je nutné dva vstupy sdílet (pokud to logická funkce dovoluje). Na druhou stranu může menší buňka vést k lepší využitelnosti logiky, podobně jak jsme naznačovali u jemnozrnných struktur. V nejnovějších FPGA obvodech Xilinx Spartan-6 (obvody střední třídy) se již ani pojem logická buňka neuvádí, mluví se o zde o několika variantách tzv. **řezů** (*slice*, dříve Xilinx tímto pojmem označoval spojení dvou logických buněk). Řez se zde skládá ze čtyř 6vstupových LUTů a osmi klopných obvodů, dva řezy pak tvoří CLB. Z vhodných variant řezů lze tak v jednom CLB vytvořit např. distribuovanou RAM o velikosti 256 bitů a posuvný registr 128 bitů [8].

2.1.2 Propojovací sítě

Propojovací sítě jsou velmi významnou částí struktury FPGA obvodu a slouží k propojení signálů mezi logickými bloky. Tvoří je několik metalických vrstev připomínajících vrstvy klasických ASIC, které se liší rozpětím základního prvku vodiče. V podstatě na tyto vrstvy máme dva protichůdné požadavky – jednak by měly být flexibilní (to znamená segmentované vodiče s velkým počtem konfiguračních přepínačů), a jednak by měly mít malá průchozí zpoždění. To lze obtížně splnit jedním druhem propojovacích sítí, a proto jich používáme několik typů. V zásadě je rozdělujeme na dvě hlavní skupiny:

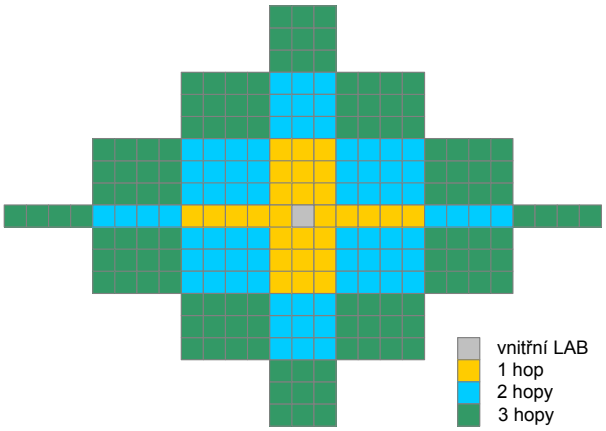
- *všeobecné propojovací sítě* pro standardní logické signály,
- *sítě pro rozvod hodinových signálů*.

Všeobecné propojení má většinou podobu mřížky, hrany této mřížky tvoří propojovací vodiče, v místech křížení jsou rozmístěny programovatelné spínače, které dovolují spojit každý vývod (hranu) s více (ne nutně všemi) vodiči. Hrany mřížky nemají jednotnou délku a lze je členit hierarchicky podle délky jejich základních segmentů zhruba do tří až čtyř kategorií. Nejkratší vodiče propojují pouze sousední bloky – označují se jako přímé (direct, single lines). Dále následují 1-2 úrovně vodičů dvojnásobné až šestinásobné délky (podle konkrétní architektury či výrobce) s rozpětím dvou až šesti sloupců nebo řádků a nakonec dlouhé vodiče, které obsáhnou 12 až 20 logických bloků nebo celý řádek či sloupec. Délky propojovacích segmentů mohou být v horizontálním a vertikálním směru různé. Na obr. 7 [9]



Obr. 7: Příklad propojení u FPGA obvodu řady Spartan firmy Xilinx [9]

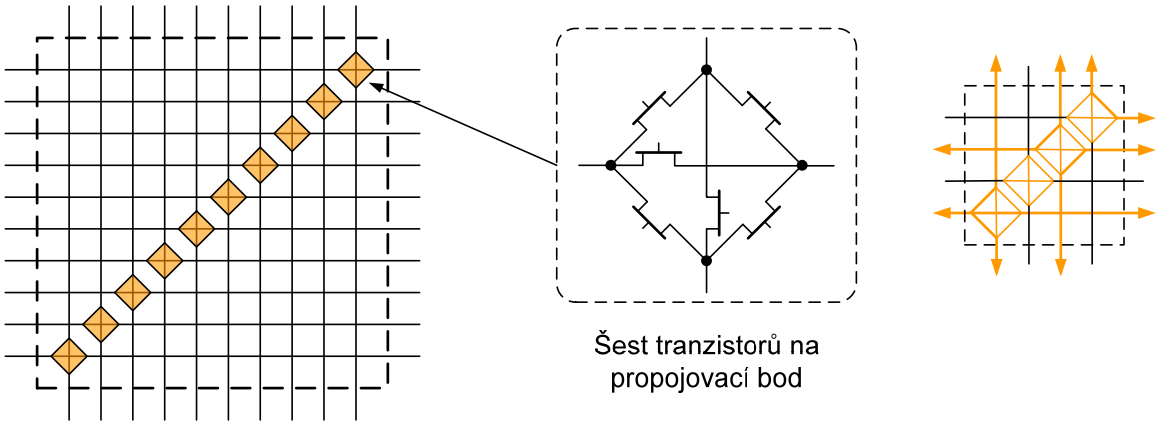
je pro ilustraci příklad propojení mezi segmenty různé délky u FPGA obvodu řady Spartan firmy Xilinx (jednotlivé úrovně segmentů jsou zde označovány Single, Double a Long). Kdyby každý propojovací vodič procházel přepínačem v každém uzlu mřížky, delší spoje by měly neúnosné zpoždění. Popisované rozdílné délky segmentů umožňují snížit počet propojovacích bodů v cestě signálu, a tím zlepšit dynamické vlastnosti. V souvislosti



Obr. 8: Dostupnost propojení mezi sousedícími LAB v FPGA firmy Altera

s počtem průchodů přes propojovací body mezi dvěma komunikujícími logickými bloky mluvíme o tzv. *hopech*. Obr. 8 ukazuje dostupnost okolních LAB u FPGA Altera Stratix II [5]; dostupnost okolních buněk je rozdílná jak u různých výrobců, tak u různých produktových řad stejného výrobce.

Programovatelné přepínače slouží buď k připojení vstupů a výstupů logických bloků k propojovacím vodičům (označují se **PIP** – Programmable Interconnection Point, programovatelné připojovací body), nebo k propojení jednotlivých segmentů propojovacích vodičů (označují se **SMIP** – Switching Matrix Interconnection Point, propojovací body matice přepínačů). K sobě navzájem náležející SMIP se sdružují v maticích přepínačů umístěných v křížení horizontálních a vertikálních kanálů (obr. 9) – nejčastěji se označují **PSM** (Programmable Switch Matrix, programovatelné přepínací matice). Architektura matic přepínačů určuje, jak lze mezi sebou spojovat segmenty vodičů ležící v sousedních kanálech. Nejčastěji se vyskytuje *disjunktní architektura*, v níž lze propojit pouze sousední segmenty náležející do téže kategorie (pořadové číslo vodiče v kanálu), takže propojovací vodiče jsou rozděleny do několika domén, které mezi sebou nelze propojit. Pokročilejší architektury zavádějí diagonální přepínače, čímž se zmenšuje počet nepropojitelných domén (např. *univerzální* nebo *Wiltonova architektura*) [10].



Obr. 9: Princip a možnosti programovatelné přepínací matice PSM firmy Xilinx

Zpoždění v propojovacích cestách je v FPGA obvodech značné a může dosáhnout i několikanásobku zpoždění v logických obvodech. Dokud návrhový systém neprovede rozmístění logiky na jednotlivé buňky, lze zpoždění ve vodičích obtížně odhadovat.

Technologie spínání se liší a záleží hlavně na tom, zda je spínací prvek aktivní (např. logické výhybky, přepínací multiplexory) nebo je pasivní (v závislosti na použité technologii např. přenosová hradla nebo jednorázově přerušitelné propojky – více v kap. 2.4). Obecně jsou pasivní spínače významným zdrojem zpoždění. Vnášejí do cesty signálu jak odpor, který je řádově větší než je odpor přilehlého úseku vodiče, tak příčnou parazitní kapacitu. K tomuto je třeba připočítat i výstupní odpor logických členů a vstupní kapacitu hradel. Uvážíme-li, že v cestě signálu je větší počet spínačů a zároveň, že dochází k větvení signálu, je výpočet či

odhad zpoždění problematický. Zpoždění v dlouhých propojovacích cestách lze snížit rozdělením cest na kratší úseky, které oddělíme budiči (zesilovači). Tyto vložené budiče zajistí vyšší proud pro nabíjení/vybíjení parazitních kapacit následných spojovacích cest a vstupů hradel, a tak přenos signálů zrychlí.

Sítě pro rozvod hodinových signálů

Nejvíce kritické na časové zpoždění v propojovacích vodičích jsou hodinové signály. S rostoucí rychlostí signálů a se zvětšujícími se rozměry čipu roste nebezpečí fázových posunů. Proto jsou v FPGA obvodech pro hodinový signál vyhrazeny zvláštní rozvody s žádnou nebo velmi omezenou programovatelností a s pečlivě navrženou topologií. Samozřejmě takového speciální sítě zabírají nezanedbatelnou plochu čipu, a proto jich nebývá na čipu příliš mnoho (řádově jednotky až desítky). Tyto rozvody se nepoužívají pouze pro hodinové signály, ale i pro jiné signály, u kterých je požadováno velké větvení signálu, malé časové zpoždění či minimální skluz (např. pro signály set/reset nebo CE). Hodinové rozvody na čipu se dále ještě většinou člení na:

- *globální rozvody* – procházejí po celém čipu.
- *regionální rozvody* – pokrývají pouze určitý region čipu. Plocha čipu bývá rozdělena na 6 až 18 regionů (konkrétně u řady Xilinx Virtex-6) a v každém regionu je několik (řádově jednotky) těchto rozvodů.
- *rozvody hodin pro vstupně-výstupní buňky* – tyto rozvody jsou navrženy pro zvláště rychlé signály a slouží pro serializaci/deserializaci vstupních či výstupních toků dat.

Návrh hodinových rozvodů je pro návrháře transparentní, je nutno však dodržovat zásady korektního popisu v HDL jazycích. Návrhový systém automaticky vkládá do rozvodů budicí zesilovače (budiče, buffery), které zajistí potřebnou kvalitu signálu, pokud větvení signálu přesáhne určitou mez. Vstup do hodinových rozvodů je možný jak ze speciálních k tomu určených pinů FPGA, tak i z vnitřní globální propojovací sítě. K problematice rozvodu hodinových signálů se ještě vrátíme v souvislosti s obvody pro úpravu hodinových signálů (kap. 2.2.3).

2.1.3 I/O bloky

Významnou částí všech FPGA obvodů jsou vstupně-výstupní bloky, které zajišťují tok dat mezi vnitřní logikou a I/O piny; přizpůsobují logické úrovně vně a uvnitř čipu, zesilují výstupní signály. Každému standardnímu I/O pinu přísluší právě jedna buňka, která může pracovat jako vstupní, výstupní nebo obousměrná.

Součástí I/O bloků jsou i struktury pro podporu různých vstupně-výstupních *napěťových standardů*, a to jak s jednou polaritou (single ended), tak diferenční. Každý ze standardů

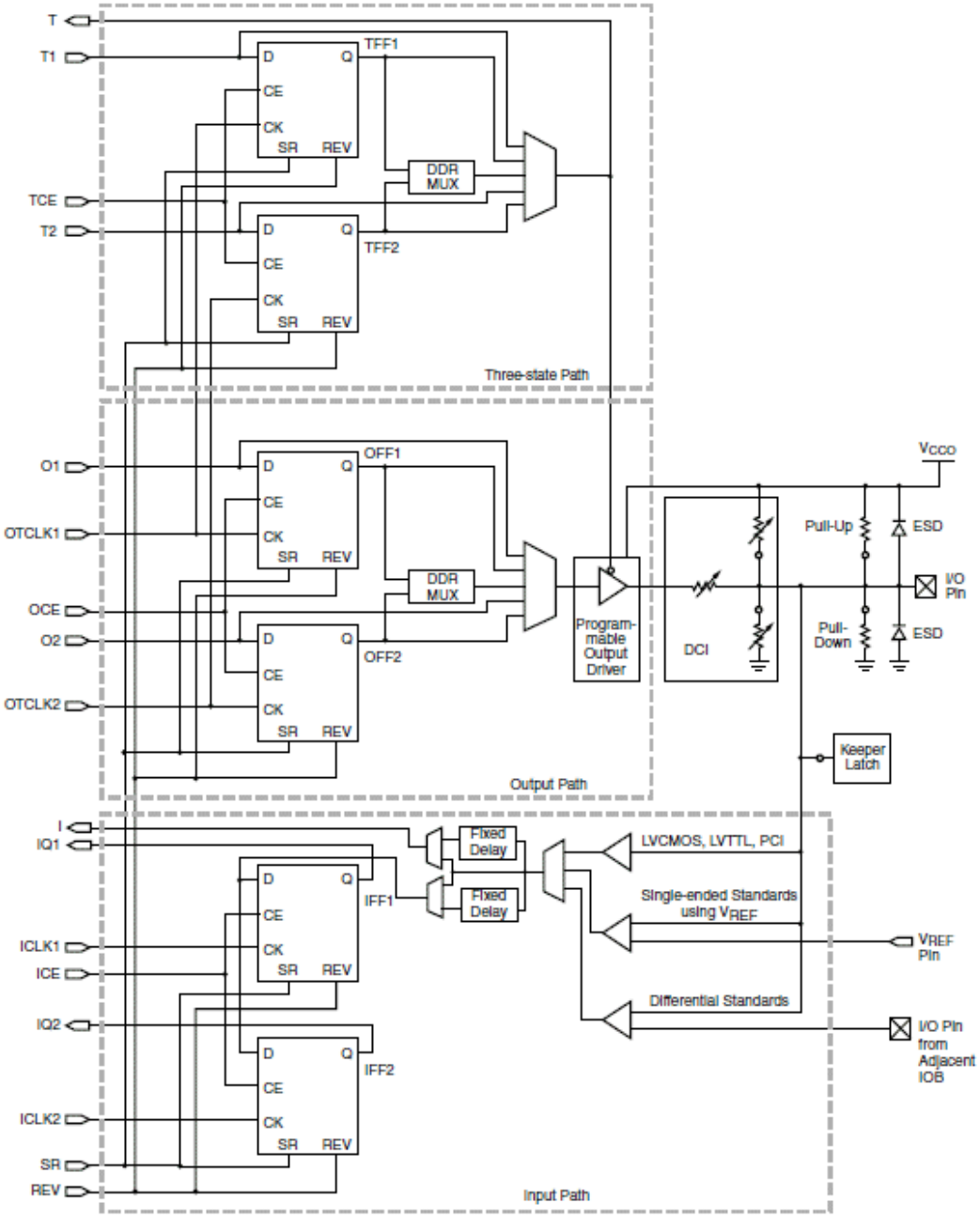
v podstatě vyžaduje jiné napájecí napětí I/O buněk (pohybuje se typicky v rozmezí od 1,2 V do 3,3 V). Aby mohl čip komunikovat s více různými standardy, rozdělují se buňky do tzv. **bank**. Každá banka pak může být připojena na jiný napájecí zdroj. V jedné bance mohou být kombinovány vstupy a výstupy různých standardů (LVCMOS, LVTTL, LVDS, atd.), ale pouze na stejném napájecím napětí. Počet bank je závislý na velikosti FPGA obvodu a pohybuje se od 8 do 30 (u řady Xilinx Virtex-6) [11]. Rovněž počet I/O pinů v jedné bance závisí na velikosti obvodu a u největších hradlových polí dosahuje až 48 pinů (u řady Altera Stratix IV) [12].

Ze standardů s jednou polaritou jsou nejčastěji podporovány LVTTL, LVCMOS, PCI, HSTL a SSTL, z diferenčních LVDS, HyperTransport, LVPECL, BLVDS, diferenční HSTL a SSTL. Některé z uvedených standardů jsou podporovány ve více napěťových úrovních. U diferenčních standardů je třeba sdílet vždy dvě I/O buňky pro jeden signál (kanál). Přenosové rychlosti na I/O pinech mohou dosahovat stovky Mb/s, u nejmodernějších obvodů dosahují diferenční standardy rychlosti až 1050 Mb/s (řada Xilinx Spartan-6) [13].

Vlastní obvodové zapojení není jednoduché, pro názornost byla vybrána I/O buňka FPGA obvodu řady Xilinx Spartan-3 na obr. 10 [14]. Na tomto relativně složitém zapojení se pokusíme demonstrovat velkou variabilitu této buňky, a přitom její strukturu zobecnit. Struktura I/O buňky obsahuje 3 základní signálové cesty:

- vstupní cesta – přenáší data z pinu do vnitřní logiky,
- výstupní cesta – zajišťuje přenos dat z vnitřní logiky na výstupní pin,
- cesta ovládající třístavový výstup (stav vysoké impedance).

Ve všech cestách jsou multiplexory sloužící k přepínání signálů do různých linek či obvodů, příp. umožňující inverzi signálu nebo zařazení určitého zpoždění do cesty signálu. Programovatelné zpoždění lze zařadit jak u vstupního, tak u výstupního signálu, a podle druhu FPGA obvodu může být pevné nebo proměnné. Jeho úkolem je zlepšení časování při zápisu dat do registru (eliminace změny dat v okamžiku příchodu aktivní hrany hodinového signálu). Klopné obvody použité v I/O buňkách mají programovatelné vlastnosti – lze např. volit řízení hranou nebo hladinou, aktivace vstupu CE (Clock Enable), různé varianty synchronního nebo asynchronního resetu či setu. Dvojice klopných obvodů ve všech třech cestách slouží převážně pro podporu DDR (Double Data Rate) přenosu u pamětí, kde se pro zdvojnásobení přenosové rychlosti používá náběžné i sestupné hrany hodinového signálu. Speciální dvouhranové klopné obvody se v architekturách FPGA nevyskytují a řeší se použitím dvou standardních klopných obvodů – jeden KO reaguje na náběžnou hranu, druhý na sestupnou a jejich výstupy střídavě přepíná DDR multiplexor (více kap. 4.6). Dva klopné obvody ve vstupní cestě je možné také použít k realizaci dvojitého synchronizátoru pro odstranění metastabilit (viz kap. 4.5.1).



Obr. 10: Vnitřní zapojení I/O buňky obvodu Spartan-3 firmy Xilinx [14]

Na vstupně-výstupním pinu lze volitelně naprogramovat pull-up a pull-down rezistory. Dále každý pin obsahuje dvě diody pro ochranu proti kladnému i zápornému přepětí (zhruba většímu než 0,7 V). Je zde také blok DCI (Digitally Controlled Impedance), který umožňuje ke každému pinu vložit zakončovací rezistor, což zjednodušuje konstrukci desek plošných spojů a eliminuje nežádoucí odrazy signálů. Firma Altera označuje podobný blok zkratkou OCT (On-Chip Termination). Napájecí úroveň pinu včetně obvodu DCI souvisí s přiřazením do příslušné banky, jak již bylo naznačeno.

Ve výstupní cestě je zařazen programovatelný výstupní driver, který má v podstatě tři úkoly:

- umožňuje měnit *rychlost přeběhu* (slew rate) – nejčastěji ve dvou až třech stupních, např. rychlá (Fast) – střední (Medium) – pomalá (Slow). Snižování rychlosti signálů omezuje velikost proudových nárazů, které mohou být příčinou vzniku rušení, zejména jsou-li výstupy zatíženy zátěží s větší kapacitou (např. u sběrnic) nebo dochází-li k současnému přepínání více výstupních vývodů. Snižování rychlosti přeběhu je ovšem doprovázeno přídavným zpožděním výstupního signálu a má význam pouze při vyšších výstupních proudech (nad 8 mA). Implicitně bývá rychlost přeběhu nastavena nejvyšší hodnotu (Fast).
- určuje *výstupní proudové zatížení* – je programovatelné nejčastěji v rozmezí od 2 do 25 mA. Výstupní proud závisí na zvoleném napěťovém standardu, a souvisí s ním i výkonnost a kvalita výstupního signálu (maximální frekvence, rychlost přeběhu, napěťové špičky, šum apod.).
- umožňuje nastavení výstupní cesty do *stavu vysoké impedance*. Tato funkce má význam nejen při nastavení I/O pinu jako vstupního, ale také v režimu výstupu jako otevřeného kolektoru nebo funkci OE (Output Enable).

Další podporovanou vlastností I/O buňky bývá tzv. *obvod přidržení úrovně* (bus hold, někdy označovaný keeper nebo aktivní terminátor), který eliminuje potřebu definovat vnějším zapojením platnou logickou úroveň na vývodech, které nejsou právě v aktivní výstupní funkci. Tento obvod přidržuje vývod v platném logickém stavu, do něhož byl naposledy aktivně vybuzen (na principu bistabilního klopného obvodu). To je výhodné zejména při použití výstupů obvodu jako třístavových budičů sběrnic, které při uvedení všech budicích zesilovačů do stavu vysoké impedance mají zajištěnou platnou logickou úroveň.

2.2 Speciální strukturní prvky

V předchozí kapitole jsme probrali základní (nutné) prvky FPGA obvodů. Trendem posledních let je doplňování čipu FPGA o některé komplexní podpůrné obvody a obvodové celky. Některé z těchto obvodů by bylo možné realizovat i v běžných univerzálních logických blocích (paměti, násobičky, sčítačky, procesory aj.). Při této realizaci by se však spotřebovalo mnoho obecných strukturních prvků a výsledný celek by nebyl příliš efektivní jak z hlediska zabrané plochy čipu (logiky), tak z hlediska dynamických vlastností. FPGA jednotlivých výrobců, ale i produktové řady jednoho výrobce, se od sebe výrazně odlišují právě touto „přidanou hodnotou“. Vlastní aplikace speciálních struktur je jednoduchá - buďto návrhový nástroj při syntéze z RTL úrovně sám rozpozná vhodnost jejich použití nebo se vloží jako samostatná komponenta (makro) do HDL návrhu. Podívejme se na nejpoužívanější vkládané struktury.

2.2.1 Statická paměť RAM

Paměťové buňky jsou dnes velmi používaným prvkem všech rozsáhlejších číslicových systémů. Používají se v nejrůznějších velikostech, od nejmenších realizovaných jednotlivými klopnými obvody, přes skupiny těchto obvodů sdružených v registrech různé datové šířky, až k dvourozměrným či případně vícerozměrným paměťovým strukturám. Složitější struktury jsou organizovány nejčastěji jako jednoportové či dvouportové paměti, které mohou být vybaveny dalšími řídicími obvody pro realizaci FIFO, LIFO apod. Vytváření paměťových buněk ze standardní logiky je neefektivní a pomalé, rovněž používání externích pamětí není také ideální. Proto jsou v moderních architekturách FPGA implementovány speciální struktury sloužící právě pro efektivní realizace pamětí. Výhodou takového řešení jsou dobré dynamické parametry a úspora místa oproti běžným vnitřním registrům, kompaktnost a snadná modifikace oproti vnějším pamětem. Paměťové prvky v FPGA obvodech je možné realizovat třemi způsoby [15]:

- speciální struktury pro rozsáhlejší paměti, tzv. **bloková paměť RAM**,
- tabulkami LUT v logických buňkách, které je možno nakonfigurovat jako paměť RAM, tzv. **distribuovaná paměť RAM**,
- **klopnými obvody** v logických buňkách.

Syntetizéry návrhových systémů většinou sami dokážou vybrat nejvhodnější typ paměti (při správném HDL popisu), případně lze syntetizátor ovlivnit vhodným nastavením. Při požadavku na realizaci velmi rozsáhlých pamětí je vhodné k obvodu FPGA připojit vnější paměťové obvody. Pro tyto účely je v možnostech I/O buněk připravena podpora pro používané standardy, zejména varianty DDR pamětí.

Bloková paměť RAM

Tato vložená (*embedded*) paměť je speciálně určená právě (a pouze) k realizaci SRAM, a to zejména synchronní. Podle konkrétního výrobce FPGA obvodů se také obvykle označuje jako BRAM (Block RAM) nebo EBR (Embedded Block RAM) jednotka. Může být konfigurována jako klasická jednoportová paměť (single port RAM) nebo jako dvouportová (Dual Port RAM). Ve dvouportové konfiguraci bývá každá brána plně samostatná včetně adresovacích i hodinových vstupů a datových vstupů/výstupů. Řízení dvouportových pamětí může být citlivé na některé konfliktní situace, které je třeba ošetřit - zápis z obou bran do stejného paměťového místa nebo čtení a zápis z/do stejné buňky (na stejnou adresu). Bloková paměť je většinou rozdělena do bloků po celém čipu FPGA obvodu nebo do několika řad (kanálů). V FPGA obvodech bývají většinou všechny bloky stejné velikosti (např. 9, 18 nebo 36 kbitů), ale někdy jsou kombinovány s bloky většími (např. 144 kbitů). Velikosti jsou vesměs násobkem 9 kvůli možnosti použít paritní bity, resp. ECC zabezpečení. Každý blok paměti je možné konfigurovat pro různé šířky slov - např. 18 kb bloky lze organizovat jako

(hloubka x šířka) 16k x 1, 8k x 2, 4k x 4, 2k x 8, 1k x 16 nebo 512 x 32 bitů (neuvažujeme paritní bity). S hloubkou paměti souvisí pak počet adresových bitů. Bloková paměť RAM je vhodná pro středně velké paměťové subsystémy se synchronním zápisem a čtením. U těchto aplikací lze pak také dosáhnout nejlepších dynamických vlastností blokových pamětí. Při inicializaci FPGA obvodu (po připojení napájecího napětí) je možné specifikovat i počáteční obsah blokové paměti. Celkové velikosti blokové paměti se v současnosti pohybují od stovek kbitů u menších FPGA až po 38 Mbitů (u Xilinx Virtex-6) [11].

Některé vlastnosti blokové paměti jsou odlišné od vlastností běžných pamětí vyráběných jako samostatné integrované obvody, a proto je zde stručně popišme. Počáteční obsah paměti po připojení napájecího napětí je možno specifikovat v popisu HDL jazyka (případně načíst z externího souboru), takže paměť může být použita i jako paměť ROM. Toho se např. využívá při implementaci procesorů do FPGA. Odlišně se chová vstup EN (Enable), kdy při jeho neaktivním stavu je výstup paměti neměnný (nikoliv ve stavu vysoké impedance, jak je obvyklé běžné u běžných pamětí) a není možný zápis do paměti ani vynulování výstupního bufferu (paměť nereaguje na hodinový vstup) [16]. Vstup RST (Reset) je synchronní a nuluje obsah výstupního bufferu, neovlivňuje obsah vlastní paměti. Vstup WE (Write Enable) povoluje zápis dat ze vstupu DI (Data Input) do paměťového místa určeného adresou (při aktivní hraně hodinového signálu) a zapsaná data se současně objeví na výstupu DO (Data Output). Není-li signál WE aktivní, pak s aktivní hranou hodinového signálu se pouze aktualizuje výstupní buffer paměti podle hodnoty signálu na adresovém vstupu.

Distribuovaná paměť RAM

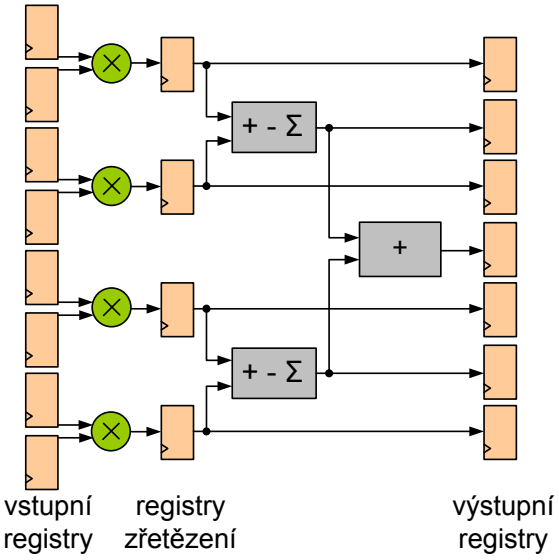
Jak již bylo naznačeno v kapitole 2.1.1, není-li tabulka LUT využívána ve své původní funkci (pro realizaci kombinační logiky), může být (pokud to umožňuje) nakonfigurována do funkce paměti RAM. Tento typ paměti se označuje jako distribuovaná paměť. Při jejím použití se tedy připravujeme o využitelné prostředky pro realizaci ostatní logiky. Distribuovaná paměť, podobně jako bloková, může být konfigurována jako jednoportová nebo dvouportová. Zápis je synchronně řízený aktivní hranou hodinového signálu, čtení je pouze asynchronní. Ze standardního 4vstupového LUTu lze získat paměť o velikosti 16 bitů (2^4). Dvouportová paměťová buňka vznikne spojením dvou běžných buněk a funkčně je spíše pseudo-dual-portová (např. jedna brána slouží pro zápis i čtení, druhá pouze pro čtení, mají společný hodinový vstup). Distribuovaná paměť RAM je vhodná pro takové aplikace, kde jde o nepříliš velké paměti těsně svázané s další logikou, jako jsou posuvné registry, čítače nebo paměti s požadavkem asynchronního čtení. Rovněž při realizaci stavových automatů implementujeme přechodovou a výstupní funkci (kombinační funkce) v podstatě ve formě paměti, čímž vlastně využíváme zmiňovanou distribuovanou paměť.

Porovnání implementace SRAM

Na základě výše uvedeného popisu možnosti realizace statické paměti RAM bylo provedeno srovnání implementace typické pseudo-dvouportové paměti 2 kbity v organizaci 256 x 8 bitů s jednou bránou pro zápis a s druhou bránou pro čtení. Popis paměti v jazyce VHDL je uveden v příloze. Byly vyzkoušeny dvě varianty – jednak klasický popis s využitím procesů a jednak popis pomocí IP jádra. Při klasickém popisu návrhový systém automaticky rozpoznal realizaci RAM a k implementaci využil pouze blokovou RAM – přesně 2048 bitů bez jakékoli další podpůrné logiky. Pokud bylo v návrhovém systému zakázáno použití blokové RAM, zmiňovaná paměť zabrala v FPGA řady Cyclone III celkem 2672 logických elementů (z toho 2056 registrů – 2048 bitů pro paměťové buňky a 8 bitů pro adresový registr), což není zanedbatelná část logiky hradlového pole. Při implementaci do logických buněk FPGA řady Stratix III, které obsahují adaptivní logické bloky ALM popisované v kap. 2.1.1, bylo zabráno pouze 954 ALUTů (adaptivních LUTů), počet zabraných registrů se nezměnil. Při realizaci pomocí IP jádra byly výsledky shodné – jak při implicitní realizaci v blokové paměti, tak při „vnucení“ do logických buněk. Problém by mohl nastat při realizaci některé speciální vlastnosti paměti, kterou bloková RAM nepodporuje. Příkladem by mohlo být zapojení paměti, které ke své činnosti využívá jak asynchronní reset, tak asynchronní set. Potom by návrhový systém použil k realizaci běžné logické elementy i za cenu jejich malé efektivity.

2.2.2 DSP bloky

V dnešní době je významnou aplikační oblastí FPGA obvodů právě zpracování číslicových signálů. Většina algoritmů pro toto zpracování vychází z principů velkého množství součinů a součtů (např. číslicová filtrace, FFT aj.). Pro podporu těchto operací jsou



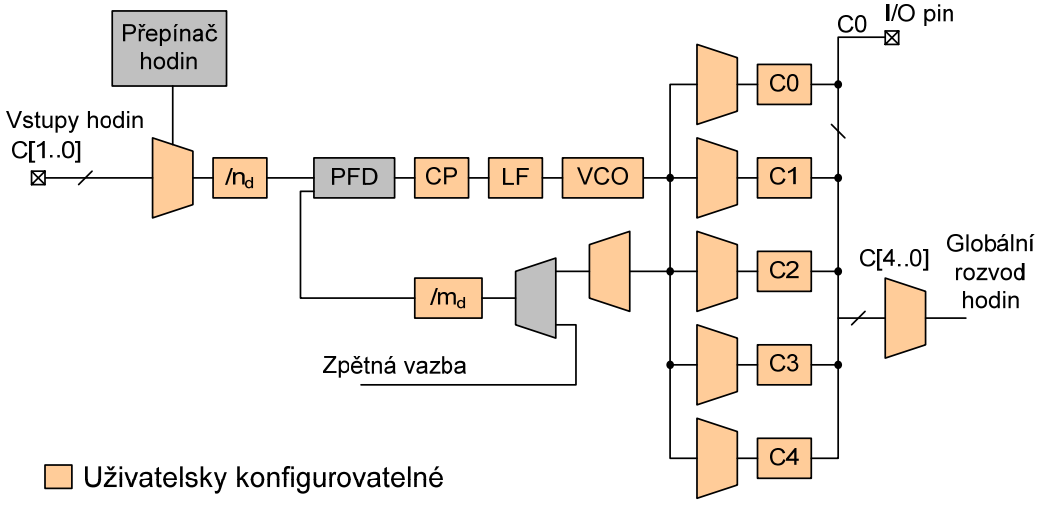
Obr. 11: Principielní schéma DSP bloku

na čipu FPGA dnes již obvyklé tzv. **DSP** (Digital Signal Processing) **bloky** (obr. 11). Základem bloku jsou násobičky doplněné sčítačkami/odečítačkami, registry a další logikou. Tyto bloky mají (oproti běžné logice) nižší zpoždění, zabírají menší plochu čipu a mají i celkově nižší příkon. Násobičky lze vzájemně zapojit do kaskády pro násobení více čísel nebo pro získání větší bitové šířky. Násobička může být také rozdělena na více částí s menší bitovou šířkou. Nejvyšší bity násobiček jsou nejpomalejší, to znamená čím méně bitů je použito, tím rychlejší je práce násobičky. Může být nakonfigurována jako kombinační nebo s registrovým výstupem (asynchronní i synchronní). Je možné zpracovávat celá čísla bez znaménka (unsigned) nebo se znaménkem (signed). Typický DSP blok obsahuje 4 hardwarové násobičky, každou se dvěma 18bitovými vstupy (18 x 18) a 36bitovým výstupem. Násobičky v bloku lze konfigurovat například jako 8 násobiček 9 x 9 bitů nebo jednu násobičku 36 x 36 bitů [17].

DSP bloky jsou také významným prvkem pro porovnávání výkonnosti, a to nejen hradlových polí – více kap. 4.3.3.

2.2.3 Obvody pro úpravu hodinových signálů

Současné FPGA obvody se již v podstatě neobejdou bez obvodů, které řídí, upravují (obnovují) a distribuují hodinový signál po celém čipu, případně násobí nebo dělí jeho frekvenci. Tyto obvody nemají jednoznačné názvy a většinou se podle výrobců FPGA označují jako DCM (Digital Clock Manager), CMT (Clock Management Tile), MMCM (Mixed-Mode Clock Manager) či sysClock. Jejich stěžejní součástí jsou fázové závěsy označované PLL (Phase Locked Loop) nebo DLL (Delay Locked Loop). Mezi hlavní funkce PLL patří eliminace časového zpoždění, vytvoření fázového posunu a digitální frekvenční syntéza (DFS, Digital Frequency Synthesis); jeho zjednodušené blokové schéma je na obr. 12 (pro názornost bylo vybráno PLL obvodu Altera Cyclone III) [18].



Obr. 12: Zjednodušené blokové schéma PLL obvodu Altera Cyclone III

Na vstupu fázového závěsu je tzv. clock switchover (přepínač hodin), který přepíná na vstup jeden ze dvou příchozích hodinových signálů. Tento signál může být pomocí děličky následně vydělen číslem n_d. Vlastní smyčka fázového závěsu se skládá z frekvenčně-fázového detektoru PFD (Phase-Frequency Detector), nábojové pumpy CP (Change Pump), filtru smyčky LF (Loop Filter), napětím řízeného oscilátoru VCO (Voltage Controlled Oscillator) a děličky ve zpětné vazbě (s dělícím poměrem m_d). U některých typů fázových závěsů v FPGA obvodech je možné použít vlastní (externí) zpětnou vazbu. Výstup PFD je funkcí jak rozdílu fází vstupních signálů, tak i rozdílu jejich frekvencí. Nábojová pumpa převádí výstupní chybové napětí PFD na náboj. Po filtraci dolní propustí přichází signál na VCO, který vytváří několik vzájemně fázově posunutých výstupních obdélníkových signálů (4 signály při posunu 90° nebo 8 signálů při posunu o 45°). Za VCO se každý signál může větvit do několika hodinových rozvodů; jsou zde multiplexory (ve funkci přepínačů) a programovatelné čítače (na obr. 12 označeny C0-C4). Tyto čítače je možné zapojovat do kaskády, a získat tak několik samostatných kmitočtů nebo signály s programovatelnou střídou. Spojením obou zmíněných funkcí lze získat několik vzájemně se nepřekrývajících hodinových signálů.

Kmitočet na výstupu VCO je dán vztahem:

$$f_{vco} = (m_d / n_d) \,.\, f_{in} \tag{2.1}$$

kde f_{in} je kmitočet vstupního signálu [Hz],
m_d je nastavená hodnota dělicího poměru děličky ve zpětné vazbě smyčky závěsu,
n_d je nastavená hodnota dělicího poměru předděličky na vstupu fázového závěsu.

Kmitočet na výstupu bloku PLL může být dále vydělen hodnotou c_d v čítačích C0-C4:

$$f_{pll} = f_{vco} / c_d = (m_d / n_d \,.\, c_d) \,.\, f_{in} \tag{2.2}$$

Další užitečnou částí PLL je *posouvač fáze* (phase shifter), který umožňuje zavést fázový posuv vstupního signálu s určitým krokem. Velikost fázového posunu je možno v průběhu činnosti posouvače zvětšovat nebo zmenšovat. Fázové posuny signálů na výstupu PLL je možné získat dvěma způsoby [18]:

- *fázový posun s jemným rozlišením* – vychází z fázově posunutých signálů na výstupu VCO (v rámci jedné periody); je dán vztahem

  $$\varphi_f = T_{vco} / k_{out} = 1 / (k_{out} \,.\, f_{vco}) = n_d / (k_{out} \,.\, m_d \,.\, f_{in}) \tag{2.3}$$

  kde T_{vco} je perioda signálu na výstupu VCO [s],
  k_{out} je počet vzájemně posunutých signálů na výstupu VCO (k_{out} = 4 při vzájemném posunu o 90°, k_{out} = 8 při posunu o 45°).

  Díky fázové smyčce lze získat řízené fázové posuny vstupního signálu s krokem např. 1/256 jeho periody (někdy řádově pouze desítky pikosekund).

- *fázový posun s hrubým rozlišením* – souvisí s násobky celých period signálu na výstupu VCO čítané v čítačích C0 až C4:

  $$\varphi_c = (c_d - 1) \,.\, T_{vco} = (c_d - 1) / f_{vco} = (c_d - 1) \,.\, n_d / (m_d \,.\, f_{in}) \qquad (2.4)$$

  kde c_d je hodnota nastavená v čítači (při $c_d = 1$ je fázový posun 0°).

Vstupní kmitočet fázových závěsů v obvodech FPGA bývá v rozsahu jednotek až stovek MHz, výstupní frekvence závisí na možnostech napětím řízeného oscilátoru VCO a může být až 1,6 GHz (řada Xilinx Virtex-6) [11]. Děličky $/m_d$ a $/n_d$ bývají pěti až 9bitové (tj. dělí od 1 do 512). Počet DCM na čipu závisí na velikosti obvodu a pohybuje se v řádu jednotek, u největších obvodů až 18 MMCM (u řady Xilinx Virtex-6) [11]. Ne všechny DCM na jednom čipu musí mít stejné možnosti – mohou se lišit v některých speciálních možnostech či dynamických parametrech.

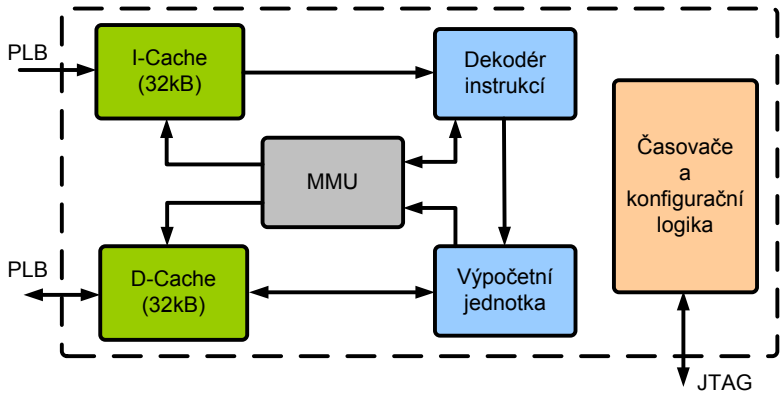
Časový management může sloužit i k blokaci hodinového signálu pro určitou část hodinových rozvodů FPGA obvodu. Tato funkce v podstatě nahrazuje hradlování signálu dodatečnou logikou (viz kap. 4.5.3); vložené zpoždění je kompenzováno přímo v DCM.

2.2.4 Procesorová jádra

Při realizaci číslicových systémů se ukazuje, že ne všechny úlohy či algoritmy je výhodné řešit speciálním paralelním hardwarem a je výhodnější použít univerzální procesorové systémy řízené programem. Dříve se tyto úlohy řešily například samostatným externím mikrořadičem. Dnes při stále se zvětšujících čipech a jejich klesajících cenách je snahou snižovat počet integrovaných obvodů pro realizaci daného systému. Proto se i procesorové jednotky stávají součástí architektury FPGA obvodů, které již v sobě obsahují i další potřebné bloky, zejména různé druhy pamětí. Návrh komplexních systémů na bázi hradlových polí a procesorů se často označuje jako tzv. **embedded design**, což se nejčastěji překládá jako „vestavěný návrh“. Procesorové jednotky obecně dělíme na dvě skupiny – *softwarové* a *hardwarové*. O softwarových procesorech se zmíníme v kap. 4.2, zde se podívejme na procesorová jádra integrovaná přímo na čip. Jejich základní konfigurace je dána výrobcem a okolní logikou je můžeme přizpůsobit požadovanému účelu.

Asi nejznámější je hardwarový procesor PowerPC vyvinutý firmou IBM, který firma Xilinx implementovala do vybraných obvodů řady Virtex-II, Virtex-4 a Virtex-5. Na obr. 13 je uvedeno blokové schéma hardwarového procesorového jádra PowerPC 440 používaného v řadě Xilinx Virtex-5 [19]. Jedná se o 32bitový superskalární procesor (umožňuje paralelní zpracovávání až dvou instrukcí) typu RISC, s registrovou ISA (Instruction Set Architecture) architekturou (32 x 32bitových všeobecných registrů). Výpočetní jednotka má 7stupňové zřetězení (pipeline), aritmeticko-logickou jednotku včetně MAC (Multiply and Accumulates) jednotky zejména pro DSP algoritmy. Dále je zde jednotka řízení paměti (MMU), dvě cache

32 kB (pro data a pro instrukce) a doplňkové periferní obvody. Komunikaci s okolím zajišťuje 128bitová sběrnice PLB (Processor Local Bus). Procesor díky své architektuře dosahuje při hodinové frekvenci 550 MHz výkonnosti až 1000 DMIPS.



Obr. 13: Blokové schéma procesorového jádra PowerPC 440

V nejnovějších trendech se aktuálně jeví perspektivnější spíše softwarové procesy, které poskytují větší flexibilitu než procesory hardwarové. Například firma Altera již delší dobu nepodporuje svůj HW procesor Excalibur, který byl implementován v řadě APEX 20KE, a také firma Xilinx v nejnovější řadě Virtex-6 již hardwarový procesor PowerPC nemá. Výkonnost hardwarových jader je však řádově dvojnásobná ve srovnání s procesorem softwarovým, a proto své místo na trhu určitě mají. Podle předběžných informací chystá firma Xilinx hardwarové IP jádro na bázi procesorů ARM Cortex.

2.2.5 Ostatní vkládané bloky

Množství vkládaných bloků v architekturách neustále narůstá, a to jak z hlediska druhů, tak z hlediska počtu jednotlivých druhů. Kromě již zmíněných typů v předchozích odstavcích bych již pouze vyjmenoval některé další, jejich výčet určitě není úplný:

- 10/100/1000 Mb/s Ethernet řadiče,
- PCI Express řadiče,
- řadiče DDR paměti,
- bloky pro podporu rychlé sériové komunikace – transceivery, aj.

Kromě výše uvedených základních i speciálních bloků jsou v architekturách další prvky, které jsou i pro běžného návrháře v podstatě skryté. Jejich užívání si řídí při syntéze většinou automaticky sám návrhový systém, případně je možné jejich vložení ovlivnit volitelnými parametry:

- serializátory/deserializátory v I/O buňkách,
- logika JTAG kompatibilní s IEEE 1149.1 / IEEE 1532, Boundary Scan budič,
- bloky pro řízení proudového odběru,

- zesilovací a třístavové budiče,
- interní oscilátor,
- blok pro dešifrování konfiguračních dat,
- start-up jednotka, aj.

O některých z těchto bloků jsme se již zmínili dříve, jiné budou ještě stručně popsány v dalších kapitolách.

2.3 Napájecí napětí

Napájení hradlových polí není jednoduchou záležitostí a je třeba mu věnovat náležitou pozornost. Velikost napájecího napětí úzce souvisí s problémem odvádění ztrátového tepla z omezené plochy čipu. Z tohoto důvodu se jej snažíme neustále snižovat. Dříve typické napájecí napětí 5 V se již téměř nepoužívá a přechází se na napětí 3,3 V a nižší. Naproti tomu je však třeba zachovat kompatibilitu s okolní logikou, kterou zajišťují I/O buňky. Snížení napájecího napětí však nepřináší jen výhody, ale vede nejen ke zhoršení dynamických parametrů, ale i ke snižování odolnosti obvodů proti elektromagnetickému rušení a nahodilým signálům (šumu).

Moderní FPGA obvody, jak již bylo popsáno, jsou v podstatě heterogenní obvody zahrnující kromě logických a I/O buněk i manažery hodinového signálu, blokovou paměť, DSP bloky, procesorová jádra apod. Některé tyto bloky vyžadují své specifické hodnoty napájecího napětí. To u hradlových polí vede obecně k potřebě většího počtu napájecích napětí. Jedno napájecí napětí, které vyhovovalo u obvodů PLD a CPLD, se postupně zvyšovalo na dvě, v současnosti tři i více (zvláště při potřebě většího počtu I/O standardů).

Typický FPGA obvod potřebuje ke své činnosti napájecí napětí, která se rozdělují do následujících tří skupin:

- *napájení vlastního jádra* s logickými bloky,
- *napájení vstupně-výstupních buněk*,
- *napájení speciálních bloků* (fázové závěsy, sériové transceivery aj.).

Proudový odběr vlastního logického jádra je většinou stěžejní položkou celkové spotřeby a u velkých FPGA obvodů se může pohybovat v řádu jednotek ampér (občas i přes 10 A) [20]. Proto velikost napájecího napětí jádra významně ovlivňuje vyzářený ztrátový výkon a zde je otázka snižování napájecího napětí nejaktuálnější. Toto snižování napětí v podstatě souvisí s použitou výrobní technologií a pohybuje se od 3,3 V (u technologie cca 350 nm), 2,5 V (220 nm), 1,8 V (150 nm), 1,5 V (130 nm), 1,2 V (90 nm), 1,0 V (65 nm) až po současných 0,9 V u největších FPGA v technologii 40 nm.

Napájení vstupně-výstupních buněk závisí na napěťových úrovních I/O standardu, pomocí kterého komunikuje obvod s okolím. Tyto napěťové úrovně se typicky pohybují v rozmezí od

1,2 V do 3,3 V. Proudový odběr I/O buněk není rozhodně zanedbatelný, ale ani u největších FPGA obvodů nepřekračuje 3 A [20]. Díky obecně vyššímu napájecímu napětí I/O buněk se podílí jejich výkonová ztráta na celkovém ztrátovém výkonu FPGA zhruba 15-30 %.

Většina FPGA obvodů obsahuje obvody **POR** (Power-On Reset), které udržují hradlové pole v resetovacím stavu, dokud nedosáhnou napěťové úrovně napájecích zdrojů stanovených hodnot. Vlastní vstupně-výstupní piny jsou během zapínání napájecího napětí z důvodu ochrany ve vysoké impedanci. Jednotlivé napájecí zdroje se většinou mohou zapínat (i vypínat) nezávisle na sobě v libovolném pořadí. U některých hradlových polí (převážně starších, bez obvodů POR) může být pořadí nabíhání zdrojů předepsáno či doporučeno – například náběh napětí I/O buněk by neměl předbíhat náběh napětí jádra. Sekvenční náběh může mít příznivý vliv na počáteční proudové špičky při nabíjení vnitřních kapacit. U některých FPGA lze v datových listech nalézt i maximální dobu náběhu (řádově jednotky milisekund). Náběh všech napájecích napětí by měl být v každém případě monotónní (bez překmitů). Nepříznivě se může projevit i přítomnost rušivých signálů (napětí) během náběhu. Proto je nutno věnovat náležitou pozornost nejen všem napájecím zdrojům, ale i napájecím rozvodům na desce plošného spoje.

2.3.1 Výkonová spotřeba FPGA

Spotřeba elektrické energie je významným faktorem při výběru hradlového pole pro konkrétní aplikaci. Spotřeba je nejen důležitá pro dimenzování napájecích zdrojů, ale také pro návrh dostatečného chlazení pro odvod ztrátového tepla. Celková spotřeba se v podstatě skládá ze dvou složek – *statické* a *dynamické*.

Statická spotřeba

Spotřeba v klidovém režimu je důležitá zejména pro stále více používané bateriové aplikace. Statická proudová spotřeba je daná součtem všech statických příkonů od jednotlivých napájecích zdrojů použitých u FPGA:

$$P_S = \sum_i U_i I_i \tag{2.5}$$

Většina výrobců má ve své nabídce nízkopříkonové řady programovatelných obvodů. Především se jedná o architektury CPLD s non-volatilním uložením konfigurace (nejčastěji na principu antipojistek nebo s pamětí flash/EEPROM – viz kap. 2.4). Mezi nejznámější řady patří Xilinx CoolRunner-II, Altera MAX IIZ a Lattice ispMACH 4000ZE. Jejich klidová spotřeba (v tzv. sleep módu) se pochybuje v rozmezí 30–50 μW [21]. Asi nejnižší spotřebu mají v současné době obvody řady Actel IGLOO nano (na principu reprogramovatelné flash technologie) s typickou spotřebou 2 μW.

Dynamická spotřeba

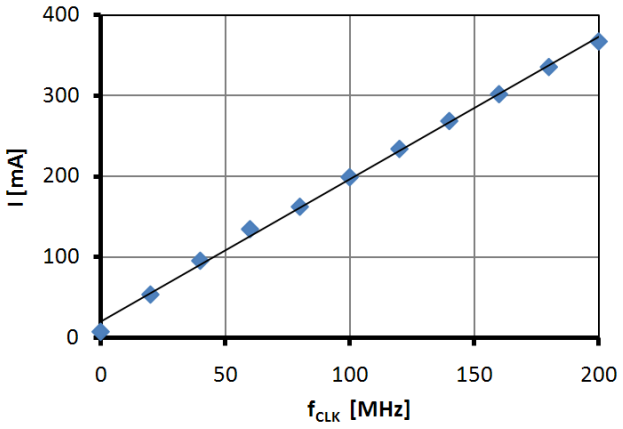
Dynamický ztrátový výkon souvisí s přechodovými ději v obvodu, příp. I/O buňkách. Největší vliv na tento výkon má (uvažujeme-li vesměs používané obvody CMOS) nabíjení a vybíjení parazitních kapacit přes příslušný unipolární tranzistor a signálový spoj. Nejčastěji se vyjadřuje idealizovaným vztahem [22]:

$$P_D = \sum_i C_i U_i^2 f_i \tag{2.6}$$

kde C_i je parazitní kapacita, U_i je napájecí napětí a f_i je pracovní frekvence. Sčítáme všechny příspěvky jak od různých napájecích napětí (nejčastěji logických a I/O buněk), tak od různých časových domén. Tento výkon lze obtížně numericky spočítat. Jeho odhad je možné provést až po kompletním dokončení samotného návrhu do hradlového pole, neboť do té doby nejsou známé konkrétní velikosti parazitních kapacit. Tyto kapacity totiž závisí nejen na výrobní technologii použitého FPGA obvodu, ale i na počtu větvení jednotlivých signálů, na délkách spojovacích cest, na počtu spínačů v dané cestě apod. Pro daný návrh v konkrétním FPGA obvodu (s daným napájecím napětím) je pak podle (2.6) závislost výkonové spotřeby na frekvenci hodinového signálu lineární. Proto se někdy udává hodnota dynamických ztrát ve W/Hz, resp. µW/MHz.

2.3.2 Zjištění konkrétní hodnoty spotřeby

Pro návrháře není zjištění konkrétních proudových odběrů z jednotlivých zdrojů jednoduché. V katalogových listech se tyto údaje téměř neuvádějí, neboť silně závisí na konkrétním implementovaném návrhu. Výrobci FPGA obvodů mají ve svých návrhových systémech zabudovány nástroje, které tuto spotřebu určí - nejznámější je XPower Estimator v systému ISE firmy Xilinx, PowerPlay Power Analyzer v systému Quartus firmy Altera nebo Power Calculator v systému ispLever firmy Lattice. Tyto nástroje jsou schopny určit nejen statickou spotřebu, ale i dynamický odběr s chybou menší než 10 % [23]. Pro výpočet



Obr. 14: Závislost napájecího proudu na frekvenci hodin

dynamického odběru je samozřejmě nutné dodat informace o taktovacích frekvencích jednotlivých časových domén případně podklady získané simulací návrhu.

Pro ověření lineární závislosti dynamické proudové spotřeby na taktovací frekvenci jsme provedli měření na obvodu Xilinx Spartan XC3S400. Jako testovací návrh jsme použili posuvný registr dlouhý 32768 bitů (zpožďovací linka). Vstupní signál pro posuvný registr byl vytvořen z hodinové frekvence připojené přes děličku dvěma (byla tak zajištěna změna logické úrovně na vstupu posuvného registru při každém hodinovém taktu). Tento návrh v obvodu XC3S400 využil 2048 tabulek LUT (z dostupných 7168) plus jeden klopný obvod na děličku (každá tabulka LUT byla nakonfigurována jako 16bitový posuvný registr). Jádro FPGA obvodu bylo napájeno z laboratorního zdroje napětí o velikosti 1,2 V. Se vzrůstající frekvencí budícího hodinového signálu spotřeba jádra podle očekávání lineárně narůstala [24] - změřená závislost je znázorněna na obr. 14. Směrnice regresní přímky odpovídá hodnotě 1,76 mA/MHz, tj. 2,11 mW/MHz. Klidová spotřeba z napájecího zdroje 2,5 V (pro pomocné bloky) byla 18 mA a ze zdroje 3,3 V (vstupně-výstupní buňky) 5 mA. Dynamická spotřeba ze zdrojů 2,5 V a 3,3 V byla i při maximální frekvenci jen o několik jednotek mA vyšší.

2.3.3 Řešení napájecích zdrojů

Všimněme si nyní, jaké možnosti má návrhář při řešení napájecích zdrojů pro FPGA obvody. Nejprve si stručně rozeberme jejich terminologii, principy a charakteristické vlastnosti.

Obecně používaná terminologie týkající se různých typů napájecích zdrojů je často nejednotná. Jako společné označení pro napěťové zdroje, které zajišťují přeměnu jednoho stejnosměrného napětí na druhé stejnosměrné napětí, použijme výraz DC-DC konvertor. Existují dva základní principy těchto konvertorů - *lineární regulátory* a *spínané regulátory*. Každý z nich má své přednosti a nedostatky, o kterých se dále zmíníme.

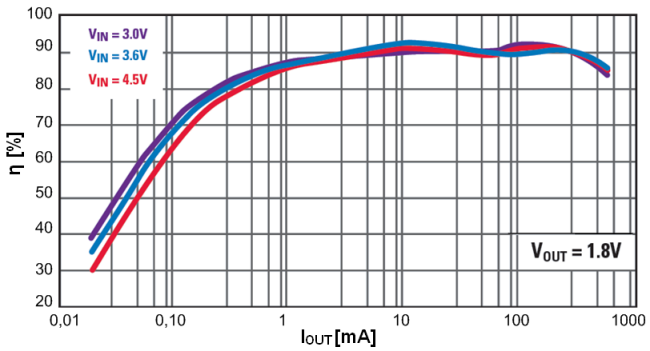
Lineární regulátory (linear regulators) jsou integrované zpětnovazební stabilizátory, které pracují se spojitým signálem. Jejich použití je pro návrháře velice snadné (až na vyhlazovací kondenzátory nevyžadují žádné doplňkové součástky). Jejich významnou předností je malé výstupní zvlnění (malý šum). Proto jsou velmi vhodné zejména pro napájení fázových závěsů, které jsou u FPGA obvodů velmi citlivé na kvalitu napájení. Vstupní napětí lineárního regulátoru U_{in} musí být vždy vyšší než požadované výstupní napětí U_{out}, což je určitá nevýhoda. Minimální rozdíl $U_{in} - U_{out}$ je označován jako „*drop-out*“ a bývá řádově desetiny voltů až 2 V. Podstatnou nevýhodou lineárních regulátorů je ale velký ztrátový výkon P_z, který na nich vzniká:

$$P_Z = (U_{in} - U_{out}) I_{out} \tag{2.7}$$

Výstupní proud I_{out} je totožný s proudem vstupním a veškerý nevyužitý vstupní výkon se mění na regulátoru v teplo. Při větším ztrátovém výkonu je nutné použít i přídavný chladič. Proto je obecně vhodnější používat lineární regulátory pro nižší napěťové rozdíly a menší proudy.

Druhou, výrazně rozmanitější skupinou napájecích obvodů, jsou tzv. **spínané regulátory** (switch regulators). Základní rozdělení těchto regulátorů je možné provést podle toho, zda jako úložiště energie používají indukčnost nebo kapacitu. Regulátory, které využívají indukčnost, bývají označovány jako *inductor based DC-DC*. Jejich nevýhodou je citlivost na kvalitu použitých součástek a jejich rozmístění na desce plošných spojů (je vhodné používat doporučené typy cívek a kondenzátory s nízkou hodnotou ESR – Equivalent Series Resistance). Nedodržení výrobcem stanovených doporučení může vést k abnormálně vysokému šumu, pískání cívky, snížení deklarované účinnosti aj. Tyto regulátory jsou běžně schopny dodávat proudy až o velikosti jednotek ampér. Výhodou bývá široké rozmezí vstupního napětí (běžně až 48 V). Spínané regulátory, které jako akumulátor energie využívají kapacity, jsou často označované jako *nábojové pumpy* (charge pumps). Složitostí zapojení se tyto obvody řadí někde mezi jednoduché lineární regulátory a poměrně komplikované regulátory s indukčností. Jejich výhodou je tedy menší prostor zabraný na desce plošných spojů, nízký počet externích součástek a poměrně jednoduchá implementace. Nevýhodou spínaných regulátorů založených na kapacitě je malý rozsah vstupního napětí (běžně do 10 V) a nižší výstupní proudy, než jakých dosahují spínané regulátory s indukčností.

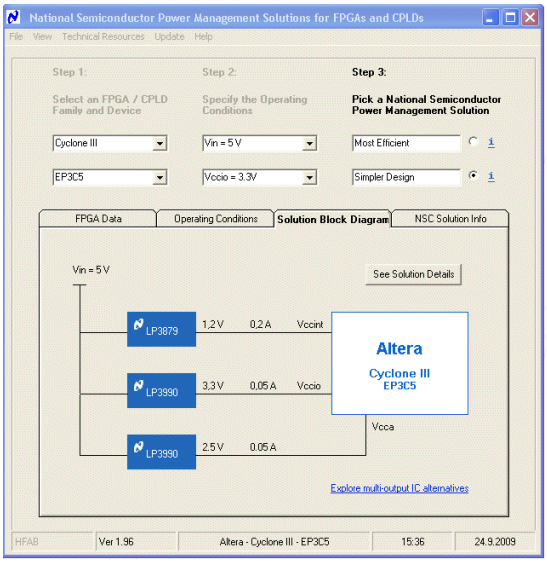
Existují dva základní módy spínaných regulátorů, tzv. **step-down** (*buck*) a **step-up** (*boost*). Step-down regulátory jsou efektivní alternativou k lineárním regulátorům, zajišťují převod vstupního napětí na nižší výstupní napětí. Jejich účinnost se běžně pohybuje od 70 % do 90 % v závislosti na typu (indukčnost/kapacita), na velikosti vstupního napětí a na konkrétním obvodu. Step-up regulátory umožňují přeměnu nižšího vstupního napětí na vyšší výstupní napětí. Účinnost těchto obvodů je podobná jako u obvodů step-down. Jako napájecí zdroje, u kterých může být výstupní napětí menší nebo větší než vstupní, se používají nejčastěji tzv. *buck-boost* obvody. Tyto obvody jsou schopné reagovat na změny velikosti vstupního napětí vůči výstupnímu a pracovat jako step-up i step-down. U většiny spínaných



Obr. 15: Závislost účinnosti na zatěžovacím proudu

regulátorů jsou implementovány alespoň některé z následujících funkcí: nastavení spínací frekvence, přechod do tzv. shutdown módu (klidový proud klesne z jednotek mA běžně na desítky μA), ochrana proti přetížení, tzv. soft-start výstupního napětí atd.

U FPGA obvodů jsou nejčastěji používány synchronní buck konvertory, které vyžadují připojení alespoň tří externích součástek – vstupního kapacitoru, induktoru a výstupního kapacitoru. Typickou závislost účinnosti těchto regulátorů na zatěžovacím proudu pro tři různá vstupní napětí ukazuje obr. 15 [25]. Někteří výrobci FPGA spolu s výrobci napájecích regulátorů dodávají speciální software, který po volbě konkrétního typu hradlového pole, jeho napájecích napětí a proudové spotřeby sám navrhne vhodné regulátory. Příkladem může být např. Power Expert, který pro FPGA firmy Altera doporučuje nejvhodnější regulátory firmy National Semiconductor (obr. 16). Při volbě „Most Efficient“ jsou preferovány spínané regulátory, při volbě „Simpler Design“ software nejčastěji doporučí lineární regulátory. Tento nástroj případně navrhne i obvodové zapojení s vnějšími součástkami.



Obr. 16: Power Expert pro volbu regulátorů u FPGA

Další možností řešení napájení je využití napájecích obvodů navržených přímo pro použití s FPGA. Příkladem může být obvod TPS75003 firmy Texas Instruments (obsahuje dva buck konvertory pro napájení jádra a vstupně-výstupních obvodů a jeden lineární regulátor pro napájení speciálních bloků) nebo obvod ISL6521 firmy Intersil (obsahuje jeden buck konvertor a tři lineární regulátory). Pokud potřebujeme řešit sekvenčnost nabíhání jednotlivých regulátorů (lineárních i spínaných) řešíme to nejčastěji pomocí vstupů „enable“, které tyto regulátory vesměs mají [26] - výstupní napětí předchozího stupně spouští stupeň následující.

Pro stále častěji používané bateriově napájené systémy je třeba minimalizovat spotřebu z jednotlivých napájecích napětí. Potom je nutné pro každou aplikaci konkrétně zvážit, jaký typ regulátorů je pro realizaci jednotlivých napětí vhodný. Účinnost lineárních regulátorů

závisí pouze na napěťovém rozdílu vstupu a výstupu, s klesajícím odběrem proudu se nemění. Oproti tomu účinnost spínaných regulátorů při nízkých zatěžovacích proudech výrazně klesá.

Návrh desky plošných spojů s FPGA

Požadavky na velikost absolutní hodnoty napájecích napětí FPGA obvodů nejsou relativně přísné – pohybují se v tolerancích cca ±5 % jmenovité hodnoty napětí. Mnohem vyšší nároky jsou na tato napětí kladeny z hlediska kolísání vlivem různých druhů rušení (zvlnění, šumy, přeslechy apod.). Proto je třeba věnovat pozornost nejen výběru vhodného typu regulátoru, ale také kvalitnímu návrhu desky plošného spoje. Doporučuje se používat vícevrstvých desek, které umožní vyhradit pro napájecí a zemnící potenciály zvláštní vrstvy. Samozřejmostí je použití kvalitních skupinových a filtračních kondenzátorů. Filtrační kondenzátory je třeba umísťovat v dostatečném množství co nejblíže napájecím pinům FPGA obvodu.

Nejcitlivější na kvalitu napájecího napětí jsou fázové závěsy, které jsou nejčastěji napájeny dvěma napájecími napětími – napětím pro analogovou část (typicky 2,5 V) a napětím pro digitální část (totožné s napětím jádra). Na obou napětích velice záleží a zpravidla jsou řešeny odděleně od ostatních napájecích přívodů (včetně oddělení od běžného napájení vlastního jádra). Například podle [14] by změna DC šumu na napájení 2,5 V neměla být větší než 10 mV/ms, dynamický šum by neměl být vyšší než 100 mV. Z hlediska návrhu desky plošného spoje se doporučuje izolovat napětí fázových závěsů od okolních signálů vytvořením speciální oblasti – tzv. *power island*. Do této oblasti je napětí přivedeno přes feritové jadérko a opět filtrováno soustavou filtračních kondenzátorů. S ohledem na zvýšenou citlivost pomocných bloků na kvalitu napájecího napětí bývají pro jejich napájení preferovány lineární regulátory.

Při návrhu desky plošných spojů je vyřešení problémů s napájením asi nejvýznamnější problém, ale není jediný. Zmínil bych se o trochu opomíjené zásadě přivedení hodinových signálů na rezervované speciální piny pouzdra, které zajistí přenos těchto signálů na globální hodinové rozvody. V principu je možné použití libovolného I/O pinu, ale jeho vlastnosti již nebudou optimální. Se vzrůstající frekvencí všech I/O signálů je stále aktuálnější používání diferenčních I/O standardů. Tyto standardy jednak zabírají dvojnásobný počet I/O pinů a jednak pro spojovací cesty platí přísná pravidla týkající se impedance vedení, jeho impedančního zakončení a jednotné délky spojů zapojených do sběrnice. To klade velké nároky jak na vlastního návrháře, tak především na kvalitní návrhový systém.

Zatím jsme se zmiňovali o možnosti rušení FPGA obvodu okolním prostředím, ale neměli bychom zapomínat také na případné rušení součástek na desce plošného spoje naopak od FPGA obvodu. Často bývají na těchto deskách na šumy citlivé analogově-číslicové či analogové součástky jako například A/D a D/A převodníky, zesilovače signálu a referenční

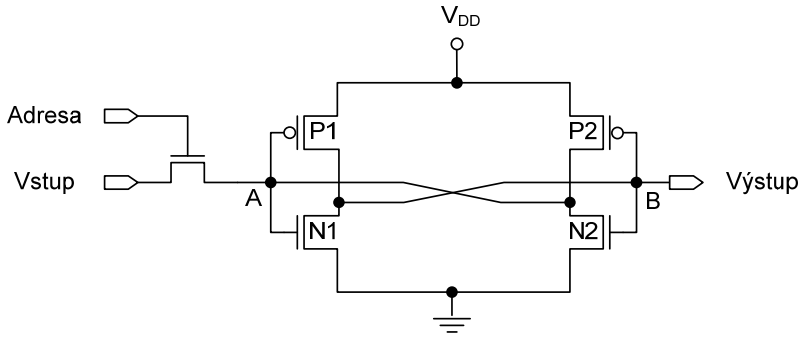
napěťové zdroje. Proto je třeba obdobné zásady, o kterých jsme psali v předchozích odstavcích, zachovávat i při aplikaci zmiňovaných obvodů.

2.4 Principy programování programovatelných obvodů

Mluvíme-li o programování, možná si spíše každý představí univerzální procesor a jeho řízení vykonáváním nějakého programu (sekvence instrukcí) uloženého v paměti. Instrukce reprezentující jednotlivé operace se v řadiči dekódují a následně vesměs sekvenčně vykonávají. Vlastní hardware CPU je v podstatě univerzální a díky konkrétnímu programu řeší danou úlohu. Sekvenční činnosti musí být samozřejmě přizpůsoben i zpracovávaný algoritmus. U zakázkových obvodů se naopak snažíme celou úlohu paralelizovat a převést jí na propojení jednotlivých funkčních bloků. Přesto i v tomto případě mluvíme o programování, tentokrát *na úrovni hardwaru*. Konfigurační program (bitstream) v tomto případě do univerzální struktury funkcí a propojení v FPGA obvodu implementuje konkrétní algoritmus. Zmiňovaný program, vytvořený kompilací v návrhovém systému, je rovněž třeba v nějakých strukturách uchovávat. Právě technologie použitá pro uložení konfigurace FPGA obvodu je významný faktor pro výběr součástky pro konkrétní aplikaci. V principu rozeznáváme dva základní typy uložení konfigurace:

- *volatilní uložení* (nestálé, závislé na zdroji elektrické energie)
- *non-volatilní uložení* (stálé, nezávislé na napájecím napětí)

FPGA s volatilní konfigurací ukládají konfigurační informace do paměťových buněk typu SRAM, což má výhodu ve snadné konfiguraci a rekonfigurovatelnosti i za běhu systému (viz níže kap. 2.7). Statická paměťová buňka pak řídí programovatelný spínač, který následně nastavuje funkci logického bloku nebo topologii propojovacích vodičů. Na obr. 17 je typické

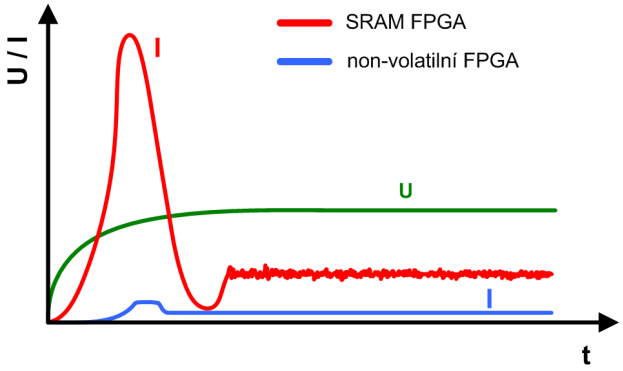


Obr. 17: Typická konfigurační buňka SRAM

zapojení konfigurační buňky SRAM používané v FPGA obvodech [27] – skládá se z pěti unipolárních tranzistorů.

Na druhou stranu použití buněk SRAM přináší i nevýhody. Programovatelný obvod musí být po startu systému nakonfigurován – obvykle z externí non-volatilní paměti (flash nebo

EEPROM), a to znamená větší potřebný prostor na desce plošného spoje a více součástek. Systém na bázi SRAM tedy není schopen pracovat ihned po zapnutí napájecího napětí, lze jej použít až po nakonfigurování, které může trvat v závislosti na velikosti obvodu a konfiguračním rozhraní od řádově milisekund do několika desetin sekundy. Vlastní konfigurační proces způsobuje po startu systému zvýšenou spotřebu elektrické energie jádra FPGA obvodu. Tato počáteční proudová špička se označuje jako tzv. *nárazový proud* (inrush current) [28] – viz obr. 18. Tento nárazový proud je třeba pro nabití vnitřních kapacit a může

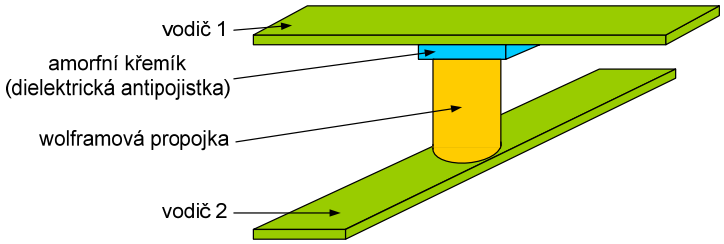


Obr. 18: Porovnání průběhu proudu a napětí při inicializaci FPGA

mít u velkých obvodů velikost v řádu jednotek ampér [26]. Na zvýšený proudový odběr je pak nutné dimenzovat i napájecí zdroj. Není-li při zapínání napájecích napětí splněna podmínka monotónnosti, může být proces konfigurace zahájen předčasně a při následném poklesu napětí potom může dojít k chybě. Nevýhodu SRAM také představuje vyšší spotřeba energie při běžné činnosti zařízení. I když je FPGA v nízkopříkonovém režimu (low power mod), odebírá stále proud ze zdroje na udržení konfigurace. U SRAM FPGA je také těžké zajistit zabezpečení intelektuálního vlastnictví proti okopírování, protože konfiguraci obvodu je v principu možné vyčíst z konfigurační paměti. Tuto nevýhodu lze dnes u většiny FPGA obvodů eliminovat použitím podpory pro šifrování konfiguračních dat (kap. 2.4.1).

FPGA s non-volatilní konfigurací používají k ukládání konfiguračních bitů nejčastěji dvě technologie – paměti *EEPROM/flash* a tzv. *antipojistky* (antifuses). Výhodou je nižší spotřeba energie výsledného navrženého systému a lepší zabezpečení intelektuálního vlastnictví (konfigurace je uložena přímo v FPGA obvodu a lze zamezit jejímu vyčtení). Další předností těchto obvodů je okamžitá použitelnost po zapnutí napájecího napětí (myšleno již po naprogramování), někdy se tyto obvody proto označují jako „*instant-on*“. Non-volatilní obvody FPGA vykazují i vyšší odolnost proti radiaci.

Antipojistka je v nenaprogramovém stavu rozpojená a programováním (pomocí zvýšeného napětí) se trvale a nevratně propojí (prorazí). Z funkce podobné opaku pojistky je odvozen i název této technologie. Realizace je založena na dielektrickém amorfním křemíku vloženém mezi dvěma kovovými vrstvami (většinou v místě křížení dvou vodičů), který je za normálního stavu nevodivý (obr. 19). Přivedením programovacího proudu (cca 5-15 mA)



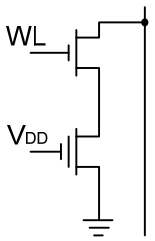
Obr. 19: Princip antipojistky

způsobí roztavení tenké vrstvy dielektrika a vytvoří se trvalý spoj. Podle výrobce mají tyto technologie různá označení - PLICE (firma Actel) nebo ViaLink (firma QuickLogic). Tato technologie dosahuje velmi dobrého poměru odporu v nevodivém a vodivém stavu – v nevodivém stavu > 100 MΩ, ve vodivém stavu od 50 Ω (ViaLink) do 500 Ω (PLICE). Tento odpor ve vodivém stavu je nižší, než dosahuje sepnutý spínací tranzistor u ostatních technologií [29], například u SRAM je to cca 0,5-2 kΩ, u EEPROM technologie 2-4 kΩ. Nízký odpor je důležitý zejména z hlediska dynamických vlastností pro zajištění rychlé změny napětí na kapacitách vodičů a hradel tranzistorů. Nevýhodou této technologie je nutnost konfigurace obvodu mimo cílovou aplikaci (v externím programátoru). Z technologického hlediska zabírají antipojistky na ploše čipu menší plochu ve srovnání s SRAM. Výhodou je také nejnižší statická spotřeba elektrické energie (i ve srovnání s EEPROM/flash) a rovněž průběh proudu po zapnutí napájení, který v podstatě nevykazuje proudový náraz (viz obr. 18).

EEPROM/flash FPGA jsou založeny na statických energeticky nezávislých přepisovatelných pamětech, které lze elektricky přepisovat či mazat. Základem je speciální unipolární tranzistor MNOS, který má mezi řídicí hradlo a substrát (kanál) vloženo další izolované (plovoucí) hradlo. Při zápisu se do tohoto hradla tuneluje elektrický náboj, který následně ovlivňuje prahové napětí tranzistoru (při čtení se tranzistor s malým prahovým napětím otevře, tranzistor s velkým prahovým napětím zůstane uzavřen). Životnost náboje na plovoucím hradle je několik let. Pro běžnou činnost (režimy programování, mazání) bývá tento tranzistor doplněn druhým unipolárním tranzistorem sloužícímu jako výběrový (viz obr. 20). Počet zápisů do jedné buňky je sice omezený, ale toto číslo se pohybuje v řádech desítek až stovek tisíc. Zvýšené programovací napětí je generováno na čipu integrovanou tzv. nábojovou pumpou, takže zvenku je paměť napájena jen běžným provozním napětím.

Paměti flash (někdy se překládají jako „mžikové“) pracují na stejném principu, rozdíl je spíše v organizaci paměti. Zatímco do klasických pamětí EEPROM se data zapisují nebo se z nich mažou po jednotlivých bytech, do flash pamětí se zapisují po celých blocích a stejně po blocích se mažou. Tím také není třeba výběrový tranzistor u každého tranzistoru paměťového. Proto jsou flash paměti celkově rychlejší a mají vyšší integraci.

FPGA obvody na bázi EEPROM/flash umožňují jak (v dnešní době velice žádoucí) programování přímo v aplikaci (tzv. **ISP**, In System Programming), tak v programátoru před vlastním použitím. Spotřeba energie je zhruba uprostřed mezi technologiemi antipojistek a SRAM.



Obr. 20: Zapojení buňky EEPROM

Na trhu FPGA existují např. i obvody kombinované – v jednom pouzdře je zaintegrována paměť flash spolu s FPGA obvodem na bázi SRAM (např. Xilinx Spartan-3AN nebo LatticeXP2). Výhodou tohoto přístupu je úspora součástek na desce plošných spojů. V tab. 1 jsou pro srovnání uvedeny klíčové vlastnosti tří zmiňovaných technologií.

Tab. 1: Srovnání vlastností konfiguračních technologií

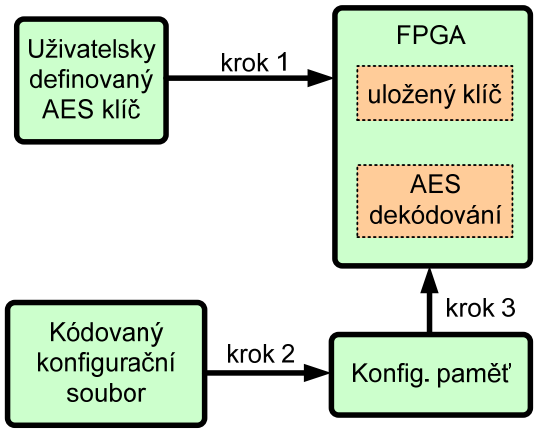
Vlastnost / Technologie	**SRAM**	**Antipojistky**	**EEPROM/flash**
Volatilní	ano	ne	ne
Reprogramovatelnost	ano (jen v systému)	ne	ano
Programovatelnost přímo v systému	ano	ne	ano
Vyžaduje konfig. soubor	ano	ne	ne
Ochrana proti kopírování	vyhovující (lze zašifrovat)	dobrá	dobrá
Velikost konfig. buňky	velká (5 tranzistorů)	velmi malá	malá (2 tranzistory)
Spotřeba el. energie	vyšší	nízká	střední
Odolnost vůči záření	horší	výborná	střední
Okamžitě použitelné	ne	ano	ano

Zajímavé řešení např. nabízí firma Lattice u FPGA LatticeXP2, který je konfigurovatelný na bázi SRAM, ale má na čipu zaintegrovanou flash paměť [30]. Tato paměť slouží sice hlavně ke konfiguraci, ale její část (tzv. FlashBAK) lze použít i k trvalému ukládání dat. Může „zrcadlit“ celou blokovou paměť a po zapnutí napájení tyto bloky inicializovat na definovaný obsah. Kromě toho má obvod možnost kdykoli během provozu zapsat zpět data z blokové paměti do flash. To lze využít např. pro ukládání kalibračních konstant či zálohování stavových proměnných nebo jiných důležitých dat, která se nesmí ztratit po vypnutí napájecího napětí.

2.4.1 Ochrana obsahu FPGA

Technologie antipojistek a EEPROM/flash jsou vesměs dobře odolné z hlediska ochrany návrhu proti tzv. zpětnému inženýrství (např. proti odcizení a následné plagiátorské výroby kopií). U EEPROM/flash FPGA lze aktivovat ochranu proti neoprávněnému vyčtení nebo nežádoucí změně (vymazání, zápis) prostřednictvím bezpečnostních bitů. Ochranu proti neoprávněnému čtení lze zrušit jen vymazáním celého obvodu. Ochranu proti zápisu (vymazání) je možno zrušit pomocí speciální programovací sekvence, jejíž náhodný nechtěný výskyt během činnosti uživatelského systému je prakticky vyloučen. U technologie antipojistek by mohli agresivnější „reverzní inženýři“ obvod rozpouzdřit a snažit se přímo analýzou čipu zjistit, které propojky jsou naprogramované. Proti tomuto jsou čipy chráněny pasivačními vrstvami, které brání opticky rozeznat naprogramované a nenaprogramované prvky. Navíc vzhledem k nízkému odporu naprogramované antipojistky dochází jen k nepatrným tepelným ztrátám, a proto ani tepelná analýza povrchu „nevede k cíli“.

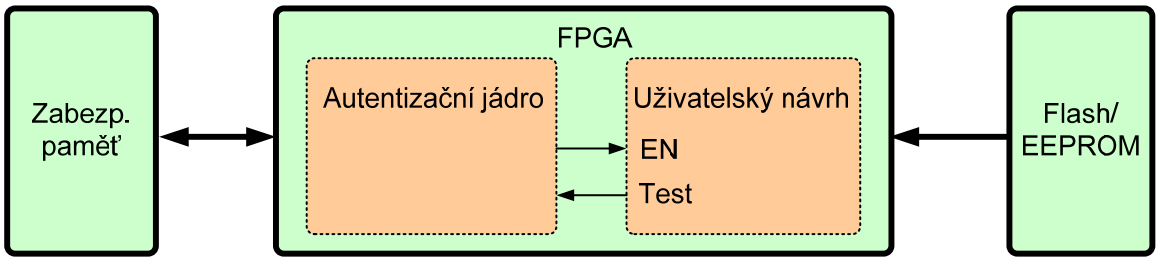
Horší je situace u FPGA založených na SRAM buňkách. Zde je třeba mít v non-volatilní paměti uložen soubor konfiguračních dat, který lze teoreticky snadno vyčíst. Určitou, vcelku dostačující, ochranou je šifrování dat, které nové FPGA obvody podporují. Používá se moderní šifrovací standard AES (Advanced Encryption Standard), který je založen na symetrickém algoritmu (používá se tentýž klíč k zašifrování i dešifrování dat). Délka AES klíče bývá 128 nebo 256 bitů [31]. Výhodou tohoto symetrického šifrování je velká rychlost a použitelnost pro velký objem dat.



Obr. 21: Princip šifrování konfiguračních dat

Zašifrování konfiguračních dat zajistí na základě dodaného AES klíče sám návrhový systém. Zašifrovaný soubor se pak standardně uloží do konfigurační paměti (většinou externí flash paměť). Návrhový systém současně umožní nahrání AES klíče do FPGA obvodu (přes JTAG rozhraní). Klíč je možné uložit buď do vnitřní volatilní paměti SRAM, která musí být zálohována externí baterií, nebo se uloží do non-volatilní paměti - typu EEPROM nebo OTP (One Time Programmable). AES klíč pak využívá blok pro on-line dešifrování konfiguračních dat, který je součástí architektury FPGA obvodu (obr. 21).

Další možností zabezpečení je používání zabezpečovacích pamětí (Secure Memory) s kryptovacím algoritmem jako je například DS28E01-100 firmy Maxim. Tato speciální paměť obsahuje blok se šifrovacím algoritmem SHA-1 (Secure Hash Algorithm), dále unikátní neměnitelný (při výrobě laserem vypálený) kód a paměťové buňky EEPROM pro tajný klíč (secret key) a další data, která nelze po naprogramování běžně vyčíst. Paměť komunikuje s FPGA obvodem přes jednovodičové rozhraní (1-Wire). FPGA obvod vygeneruje náhodná sériová data, která pošle do zabezpečovací paměti a dostane odezvu (tzv. otisk), kterou porovná se svým výpočtem. Pokud je vše v pořádku, odblokuje uživatelský návrh. Tato technologie předpokládá, že v FPGA obvodu je kromě uživatelského návrhu také speciální část zajišťující popisované funkce (generování náhodné posloupnosti, výpočet algoritmu SHA-1, obvod porovnávání otisku, musí znát tajný klíč, unikátní kód a data v zabezpečovací paměti). Popisované řešení je v podstatě použitelné ve všech FPGA obvodech a například firmy Xilinx i Altera mají volně k dispozici IP jádra, která řeší spolupráci se zabezpečovacími paměťmi a zabírají v FPGA obvodech zhruba 200 logických buněk [32], [33]. Konfigurační data pro FPGA (včetně IP jádra) jsou standardně uložena v klasické flash/EEPROM paměti (obr. 22).



Obr. 22: Principielní schéma použití zabezpečovacích pamětí

2.5 Používané výrobní technologie

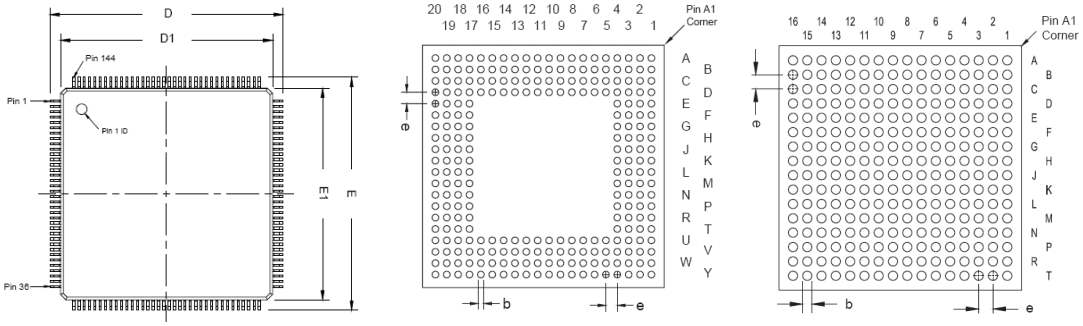
S principy programování souvisí i vlastní technologie výroby FPGA obvodů, a proto se jí stručně věnujme. Vesměs všechny druhy CPLD a FPGA obvodů jsou vyráběny unipolární CMOS technologií. Přesto nejsou u všech obvodů tyto technologie stejné a parametrově vyrovnané. V každém případě patří FPGA obvody do čela vývoje VLSI obvodů.

K výrobě FPGA s SRAM je používán standardní technologický proces CMOS bez dodatečných kroků, a proto jsou obvody SRAM vždy o minimálně jednu až dvě technologické generace napřed před jinými technologiemi FPGA. Právě obvody FPGA na bázi SRAM jsou dnes prvními návrhy vyráběnými v nových technologických procesech. V současné době jsou technologicky nejlepší obvody (Stratix IV firmy Altera, Virtex-6 firmy Xilinx) vyráběné technologií 40 nm, používají napětí jádra 0,9 V a umožňují vnitřní frekvence cca 600 MHz. Také SRAM FPGA obvody „nižších“ řad (např. Altera Cyclone III nebo Xilinx Spartan-6) jsou vyráběny špičkovějšími technologiemi (45-65 nm, napětí jádra 1 až 1,2 V).

Technologie antipojistek a EEPROM/flash vyžadují oproti standardnímu výrobnímu procesu CMOS dodatečné technologické operace, jsou tedy technologicky náročnější, a proto se při jejich výrobě nepoužívají nejnovější technologie. Proto i hustota integrace u nich je na nižší úrovni. Dnešní nejlepší technologie výroby antifuse FPGA používají technologii 1,5 μm a mají napětí jádra 1,5 V. Obdobně technologie EEPROM/flash používají nyní minimální rozměry od 0,18 do 0,35 μm s napětím jádra v rozmezí 1,8-3,3 V. Kombinované obvody LatticeXP2 [30], o kterých zde již byla zmínka, mající na čipu SRAM FPGA společně s flash pamětí jsou vyráběny nejmodernější technologií dovolující realizovat flash paměti, tj. 90 nm. Z uvedených údajů je zřejmé, proč jsou nejrozsáhlejší FPGA obvody založeny právě na principu SRAM, i když se to na první pohled může jevit nelogické vzhledem ke složení paměťové buňky z 5 tranzistorů oproti 2 tranzistorům u EEPROM/flash.

2.6 Pouzdra FPGA obvodů

Se vzrůstajícím objemem logiky integrovaným na čipu souvisí i vývoj pouzder. Pomalu se přestávají vyrábět obvody s méně než 100 vývody a počty pinů největších FPGA se blíží 2000 (konkrétně až 1932 u Altera Stratix IV [12]). Z pohledu „amatérského“ návrháře, který si chtěl sám navrhnout schéma i desku s plošnými spoji, a nakonec si vše připájet, je situace problematická. Pouzdra typu PLCC (Plastic Leaded Chip Carrier), která bylo možné vložit do patice, končí na 84 pinech, pouzdra TQFP (Thin Quad Flat Pack) na 144 pinech. Pouzdra dnešních velkých obvodů již nemají vývody po obvodu (tyto varianty končí cca u 300 pinů), ale v maticové struktuře na spodní straně čipu. Tato pouzdra se označují BGA (Ball-Grid Array) nebo FBGA (FineLine BGA) – viz obr. 23. Používají se i jiné tvary pouzder, více např. [34].



Obr. 23: Nejpoužívanější pouzdra FPGA obvodů (TQFP, BGA a FBGA) [34]

Výhodné pro návrháře je, že různé varianty CPLD/FPGA obvodů dané produktové řady konkrétního výrobce ve stejném pouzdře jsou vesměs pinově kompatibilní, a tak při potřebě větší logiky není třeba navrhovat novou desku plošných spojů, stačí jen zvolit „výkonnější“ s větším množstvím vnitřní logiky.

2.7 Rekonfigurovatelnost obsahu FPGA

Dosud jsme uvažovali využití FPGA k realizaci konvenčních aplikací – obvod jednou nakonfigurujeme, a následně používáme. Podstata FPGA obvodů ale umožňuje i poněkud netradiční použití – možnost rekonfigurovatelného hardwaru. Tuto možnost lze využít v případě, kdy vyžadujeme více implementovaných algoritmů a nepotřebujeme, aby fungovaly současně. Přehráváním funkcí částí obvodů můžeme v podstatě FPGA vícenásobně využít a k implementaci celkové funkce použít menší obvod. Další využití rekonfigurace může být např. hlídání své činnosti, a při detekci chyby se překonfigurovat tak, aby se problematické místo obvodu nepoužívalo.

Rekonfigurovatelnost však může mít mnohem obecnější význam – může v budoucnu stírat rozdíly mezi návrhem hardwaru a programováním softwaru. Kompilátor některého vyššího programovacího jazyka sám rozhodne, jakou část aplikace implementovat např. ve formě koprocesoru FPGA a jakou zkompilovat do strojového kódu procesoru [35], [36]. Celá konfigurace HW a SW takového systému by se mohla dokonce za běhu měnit.

Možnost rekonfigurace FPGA velice vítají experimentátoři nového směru, pro který se ujalo označení „*evolvable hardware*“ (do češtiny se překládá jako „vyvíjející se obvody“). Evolvable hardware jsou obvody, jejichž zapojení je subjektem evolučního procesu, který dovoluje adaptaci na úrovni technických prostředků. Tato adaptace probíhá v reálném čase v závislosti na měnících se požadavcích na činnost obvodu [37].

Rekonfigurace se využívá u obvodů FPGA na bázi technologie SRAM a rozlišujeme v podstatě dva principy:

- *statickou rekonfiguraci* (static reconfiguration)
- *dynamickou rekonfiguraci* (dynamic reconfiguration)

Statická rekonfigurace je základním typem rekonfigurace, kdy dochází ke kompletnímu přehrání konfigurační paměti, a tím ke změně celé funkce obvodu. Speciálním případem této rekonfigurace je tzv. *transparentní rekonfigurace* (transparent reconfiguration). Transparentní rekonfigurace v podstatě jen minimalizuje přerušení běhu činnosti systému při změně konfigurace. Používá se u CPLD a FPGA obvodů na principu SRAM doplněné non-volatilní konfigurační pamětí flash či EEPROM. Tento typ rekonfigurace používá např. u svých několika řad programovatelných obvodů firma Lattice [38]. Vlastní rekonfigurace probíhá ve čtyřech fázích:

- do non-volatilní paměti (nejčastěji je to interní flash zaintegrovaná na čipu, ale u vybraných typů může být i externí) se během běžné činnosti obvodu (na pozadí) nahrají nová konfigurační data - přes konfigurační rozhraní JTAG,
- I/O piny FPGA se uzamknou na definovaných úrovních,
- nová konfigurace se přehraje z paměti flash do SRAM (provede se reset obvodu),
- obnoví se nová funkce FPGA obvodu a obvod převezme řízení I/O pinů.

Jelikož je flash paměť většinou na čipu (tzv. ISF, In System Flash), přenese se její obsah do SRAM po paralelní sběrnici vysokou rychlostí; nová funkce je tedy připravena cca za 1 až 2 ms. Podobně rychle se tyto obvody chovají i po zapnutí napájení, a proto se řadí do skupiny obvodů „*instant-on*“, tj. připravené k činnosti ihned po připojení napájecího napětí (za dobu 2 ms obvykle nestačí naběhnout ani oscilátor nebo napájecí zdroj).

Dynamická rekonfigurace umožňuje za provozu (v reálném čase) modifikovat funkce části obvodu bez porušení funkce zbývajících částí. Dynamická rekonfigurace může být prováděna buď přes celý obvod FPGA (což je v podstatě transparentní rekonfigurace) nebo jen přes jeho část, pak mluvíme o *částečné* (partial) dynamické rekonfiguraci. Rekonfigurace jen části obvodu trvá i kratší dobu – přehrání konfiguračního souboru celého obvodu trvá minimálně řádově milisekundy, změna zapojení malé části se zvládne v řádech mikrosekund. Oblast, která se nemění během běhu aplikace, se nazývá *statická* a oblast, která je měněna, se nazývá *dynamická*. Ve statické oblasti obvykle bývá implementována řídicí logika – většinou hardwarový nebo softwarový procesor včetně dostatečného množství paměti pro uložení různých konfiguračních posloupností (bitstreamů). V dynamické oblasti jsou pak umisťovány moduly s konkrétními variabilními funkcemi. Velikost dynamické oblasti většinou nemůže být libovolná a je dána určitým počtem logických bloků. Komunikaci mezi statickou a dynamickou částí je třeba během procesu rekonfigurace zablokovat (většinou speciálními prvky).

Rekonfigurace je prováděna pomocí konfiguračního rozhraní obvodu FPGA, které je přístupné přes vnější vývody (JTAG interface) nebo přes vnitřní konfigurační rozhraní (u firmy Xilinx je označováno ICAP – Internal Configuration Access Port). Vnitřní rozhraní se používá v případě, že konfiguraci provádí řídicí logika ve statické části obvodu. Doba rekonfigurace je dána velikostí konfigurační bitové posloupnosti a přenosovou rychlostí mezi paměťovým systémem architektury, kde je bitstream uložen, a konfiguračním rozhraním. Konfigurační bitstreamy obsahují příkazy pro vnitřní automat konfiguračního rozhraní (např. adresa konfigurovatelné oblasti, informace o velikosti konfiguračních dat, zabezpečovací cyklický kód). V principu je možné jednu funkci nahrávat do více částí nebo více funkcí nahrávat do stejné oblasti. S růstem počtu funkcí a s počtem oblastí, kam je třeba jednu funkci nahrát, roste i počet bitstreamů uložených v paměti, protože bitstream pro jednu funkci nelze kvůli zabezpečení nahrát na jiné místo v obvodu, než pro které byl určen. Pokud chceme bitstream nahrát do jiné oblasti obvodu (relokovat), musí být příslušným způsobem pozměněn (změnit cílovou adresu konfiguračních dat a přepočítat zabezpečovací kód). Dalším důvodem, proč v některých případech nelze relokovat bitstream, je nehomogenita (neregulárnost) struktury celého obvodu FPGA. Při přehrávání dat dochází k určité latenci funkce nového modulu, se kterou je třeba počítat, příp. tento výpadek funkce vhodně ošetřit. Určitou nevýhodou se v některých aplikacích může jevit také to, že vzájemně zaměnitelné oblasti musí být kompilovány již při návrhu původního systému.

Úspěšné používání dynamické rekonfigurace vyžaduje dva předpoklady – jednak podporu hardwaru FPGA obvodů a jednak podporu návrhových prostředků, které podle našeho názoru trochu zaostávají za hardwarem. Je to dáno především velkou složitostí postupu při návrhu rekonfigurovatelného FPGA. Technologie dynamické rekonfigurace je podporována jen u některých řad hradlových polí vybraných výrobců - s částečnou dynamickou rekonfiguraci přišla v roce 2002 firma Atmel u obvodu AT94K40 FPSLIC, od roku 2004 ji podporuje také firma Xilinx u svých řad Spartan (některých) a Virtex (od verze Virtex II u všech) [39]. Zatím je převážně využívána spíše jen v akademickém prostředí a běžně se v praxi nepoužívá. V budoucnu se ale dá předpokládat její prudký nárůst.

2.8 Zpoždění signálu v programovatelných obvodech

Za povšimnutí stojí vývoj zpoždění signálu v programovatelných obvodech, jak z hlediska zpoždění ze vstupu na výstup, tak poměru zpoždění ve vlastní logice a propojovacích sítích. U obvodů PLD bylo zpoždění signálu ze vstupu na výstup obvodu u všech signálů v podstatě stejné a snadno definované. Propojovací vodiče byly v podstatě součástí architektury logických funkcí a nebylo nutné jednotlivá zpoždění blíže rozlišovat. U obvodů CPLD však již zpoždění signálů ze vstupu na výstup díky různým druhům propojovacích matic (lokální, globální) záleží na konkrétní logické funkci a jejím rozmístění v architektuře obvodů. Ještě komplikovanější je situace u obvodů FPGA, kde je bez speciálních nástrojů obtížné určit dynamické parametry navrženého systému. Proto také výrobci FPGA obvodů neuvádí v katalogových listech jejich maximální frekvence, jako jsme byli zvyklí u standardních logických obvodů či PLD obvodů. Zpoždění navrhovaných obvodů určujeme z logických simulátorů a časových analyzátorů, které jsou vesměs standardní součástí návrhových systémů různých výrobců.

U obvodů FPGA, kde jsou propojovací sítě nejrozsáhlejší, komplikují významně situaci programovatelné propojovací matice a způsobují další dodatečná zpoždění. Hlavním důvodem je množství spínacích tranzistorů v propojovacích sítích, přes které jednotlivé signály procházejí. Většina zpoždění v FPGA obvodech tedy dnes již není ve vlastní logice, jak jsme byli zvyklí z běžných logických obvodů, ale právě v propojovacích sítích. Je třeba si dále uvědomit, že s rostoucí integrací na čipu se zkracují kanály použitých unipolárních tranzistorů (uvažujeme technologii CMOS, která je dominantní). Kratší kanály znamenají při stejném napájecím napětí vyšší intenzitu elektrického pole na těchto tranzistorech a zároveň kratší dráhu pro nosiče náboje. Proto se při zmenšování rozměrů zhruba s kvadrátem (bez jakýchkoli architektonických úprav) zvyšuje rychlost funkční logiky.

Rychlost šíření signálů vodiči se na druhou stranu snižuje. Vodiče si v integrovaných obvodech musíme představit ve zjednodušené podobě jako RC články [40], přes které se musí nabíjet/vybíjet parazitní kapacity vstupních hradel unipolárních tranzistorů. Se zmenšováním

rozměrů sice klesají délky vodičů, ale zároveň se zmenšuje jejich průřez - to vede ke zvětšování jejich odporu. Zmenšováním rozměrů však příznivě působí na parazitní kapacity vodičů a zejména na kapacity hradel tranzistorů. Zmenšení průřezu kanálů spínacích unipolárních tranzistorů, které musí nabíjet a vybíjet zmiňované RC články, ve svém souhrnu vede k snížení spínací energie budicích prvků, a tím k výraznému snížení rychlosti šíření signálu po čipu.

Ve svém důsledku vede zlepšování technologií výroby IO k zrychlování funkční logiky a zároveň ke zvětšování zpoždění signálů ve vodičích. Tento poměr se neustále zhoršuje a zhruba od technologie 500 nm je dominantní zpoždění ve vodičích [41]. Zastaralé názory z diskrétních logických obvodů, že zpoždění je vesměs ve vlastní logice a zpoždění ve vodičích je možno zanedbat, již dávno neplatí. Dokonce zpoždění ve vodičích již mnohdy nelze zjednodušeně modelovat jedním RC článkem, ale pro přesnější odhady je třeba na něj pohlížet jako na vedení s rozprostřenými parametry.

2.9 Kritéria výběru FPGA obvodu

Výběr obvodu pro konkrétní aplikaci není vůbec jednoduchým úkolem a má návaznost na systémové řešení popisované v kap. 4 (i přes jeho obecnou technologickou nezávislost). Shrňme si proto kritéria, která bychom měli brát v úvahu při výběru konkrétního FPGA obvodu [35]. Mezi nejvýznamnější patří:

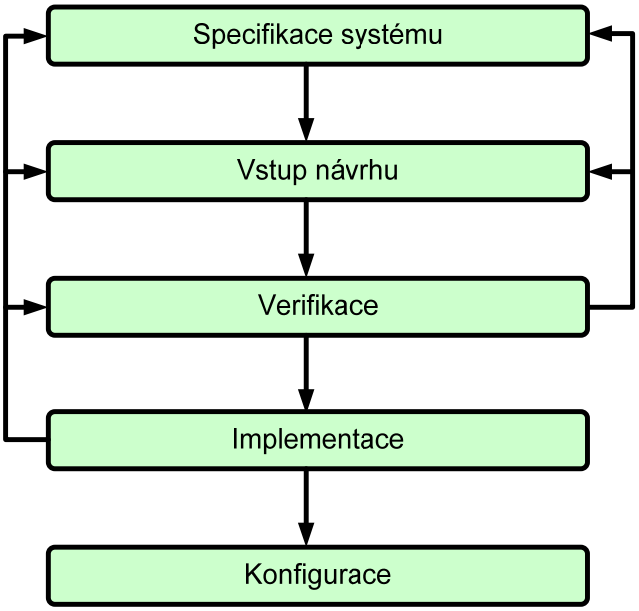
- *Volba technologie konfigurace* – zvolit SRAM, antipojistky či flash? Volba závisí především na velikosti očekávané série, výběru na trhu, cenových relacích; vliv na rozhodnutí mohou mít i očekávané pozdější změny v konfiguraci obvodu (např. změna protokolu, rozhraní apod.). Pokud potřebujeme mít obvod plně funkční ihned po zapnutí napájecího napětí (tzv. aplikace typu *start-up time*), nemůžeme zvolit SRAM.
- *Charakter operací prováděných obvodem* – bude mít spíše řídicí funkce (např. řízení komunikace) nebo bude provádět velké množství výpočtů (např. číslicové zpracování signálů)? Budeme tedy posuzovat množství speciálních bloků (pamětí, DSP bloků, řadičů Ethernet, procesorů, podpora DDR apod.).
- *Velikost obvodu* – je třeba vytvořit si alespoň hrubý odhad velikosti finálního návrhu; uvádí se nejčastěji v logických buňkách (elementech) nebo ekvivalentních hradlech. Není dobré zaplnit FPGA obvod „až na doraz“ jednak z důvodu pozdějších úprav či oprav chyb, jednak z důvodu potenciálních problémů s časováním. V téměř zaplněném obvodu je router často nucen propojovat logické prvky „oklikami“ díky vysokému stupni zaplnění propojovacích kanálů, což vede ke zvyšování zpoždění. Je rozumné nechat asi 10 % obvodu volného.

- *Počet potřebných vstupních a výstupních signálů* – s počtem I/O pinů souvisí i výběr velikosti pouzdra - vlastní počet pinů pouzdra bude výrazně větší o množství napájecích, programovacích, řídicích a jiných pinů.
- *Dynamické vlastnosti* – zejména jde o maximální rychlost hodinového signálu v logice FPGA. Většina produktových řad se vyrábí v různých rychlostech (tzv. *speed grade*). Stejné číslo speed grade u různých řad obvodů FPGA neznamená stejné zpoždění – většinou novější a větší obvody s lepší výrobní technologií jsou rychlejší.
- *Spotřeba energie* – se spotřebou energie souvisí i vyzařované ztrátové teplo během činnosti systému. Jiná je situace pokud navrhujeme nízkopříkonovou aplikaci napájenou z baterie, a jiná u velké a výkonné DSP aplikace. Příkon se s dynamikou systému zvyšuje a někdy je třeba uvažovat i o přídavném chlazení (včetně vhodného pouzdra FPGA).
- *Odolnost v prostředí finální aplikace* – musíme zvážit parametry ovlivňující zejména spolehlivost (teplotní rozsah, radiační odolnost, prostředí automobilové či družicové techniky apod.).
- *Ochrana obsahu* – souvisí se zabezpečením intelektuálního vlastnictví a s možností okopírovat navrženou konfiguraci. Non-volatilní obvody obecně poskytují vyšší úroveň zabezpečení, u obvodů na bázi SRAM je třeba zvážit použití šifrování.

Pořadí důležitosti jednotlivých kritérií závisí na aplikaci, pro kterou FPGA obvody vybíráme.

3 Metodika návrhu programovatelných obvodů

Jak již bylo naznačeno v kap. 1, minulostí jsou doby, kdy návrh integrovaných obvodů byl záležitostí pouze výrobců technologie. S rozvojem výrobních technologií, výpočetní techniky a softwarových návrhových systémů se přesunul návrh programovatelných zakázkových obvodů zcela na systémové návrháře jednotlivých specializovaných oborů, kteří přesně dokážou specifikovat své požadavky na funkci u stále rozsáhlejších a specializovanějších obvodů. Vývojové nástroje lze, zejména v různých omezených verzích, získat celkem snadno (jsou i volně k dispozici na internetu), za profesionální nástroje je naopak třeba platit i v řádech tisíců dolarů. Vlastní metodika návrhu se pro různé typy zakázkových obvodů i různé návrhové systémy liší, i když ne zásadně. Rozeberme si návrhový cyklus typický pro obvody FPGA, který se skládá ze specifikace požadavků, zachycení návrhu, verifikace a implementace, viz např. [42]. Typické etapy znázorňuje obr. 24.



Obr. 24: Metodika návrhu FPGA obvodů

3.1 Specifikace funkce

Prvním krokem návrhu musí být vždy specifikace funkce. Na jejím správném vytvoření závisí úspěch celé práce. Specifikace musí popisovat všechny požadované aspekty chování budoucího obvodu, musí být jednoznačná a bezchybná. Často na projektu spolupracuje více návrhářů, pak je třeba, zejména vzhledem k návaznosti, jeho odsouhlasení všemi participujícími stranami. Blíže je této problematice věnována kap. 4.

3.2 Vstup návrhu

Vstupní popis návrhu (*design entry*), označovaný také jako zachycení návrhu (*design capture*) je proces, při němž přenášíme svou představu o funkci a struktuře budoucího obvodu do počítačem zpracovatelné formy popisu. Specifikaci lze provádět na různých úrovních abstrakce:

- *Na hradlové úrovni* (*strukturní úroveň*) – Toto je v podstatě nejnižší úroveň abstrakce pro zadávání číslicových obvodů, výsledný obvod navrhujeme ve formě zadávání základních součástek (hradel) a jejich vzájemného propojení – nejčastěji kreslením logického schématu. Proto se tato úroveň označuje také jako schematická. Výhodou tohoto popisu je jeho srozumitelnost a jednoznačnost. Nevýhody však převažují. Logické schéma nedává žádnou volnost pro syntézu a je obtížně přenositelné do jiné architektury FPGA nebo do jiného vývojového prostředí. Vlastní proces zadávání (kreslení) je pomalý a únavný, a pro velké obvody v podstatě nepoužitelný. Obdobně časově náročné jsou úpravy či opravy. Pokud se rozhodneme výsledné schéma rozšířit (např. přidat další bit čítače), je třeba přikreslit další hradla. Snadno se můžeme dostat do situace, kdy je nutné kompletní překreslení. Chceme-li např. realizovat stavový automat, musíme nejprve navrhnout jeho přechodovou a výstupní funkci, a pak nakreslit schéma. Z uvedených důvodů se strukturní návrh používá ve velmi omezené míře pro návrh jednoduchých obvodů.
- *Na úrovni RTL (meziregistrové přenosy)* – Využívá se popis na úrovni RTL (Register Transfer Level), tj. navrhovaný obvod popisujeme jako sadu registrů propojených kombinační logikou realizující logické operace. Tato úroveň v podstatě popisuje tok dat mezi registry. Výhodou použití RTL je vyšší stupeň abstrakce, což zvyšuje produktivitu práce a zpřehledňuje vlastní návrh. Návrhář však musí řešit výběr algoritmu, jeho paralelismus, počet a velikost sběrnic, řídicí obvody i časování jednotlivých registrů. Popis obvodu je realizován v textové podobě pomocí zápisu v některém jazyku pro popis technických prostředků, tzv. HDL (Hardware Description Language). Nejčastěji se používá jazyk VHDL nebo Verilog. Jazyky HDL mají podporu pro vytváření parametrizovatelných struktur – např. čítač může mít obecnou šířku, nastavením konstanty v kódu zvolíme jeho konkrétní velikost. To umožňuje znovu použít jednou navržené bloky. Vyšší, technologicky nezávislá, úroveň abstrakce výrazně zvyšuje přenositelnost mezi různými technologiemi. Popis na úrovni RTL je dnes nejpoužívanější pro zachycení návrhu synchronních sekvenčních systémů.
- *Na algoritmické úrovni* (*behavioristická úroveň*) – Popis na úrovni algoritmu; chování celých funkčních bloků je zpravidla popsáno v některém z jazyků na „vyšší úrovni“. Použití jazyků VHDL či Verilolog není pro tuto úroveň typické,

v současnosti se preferuje např. prostředí Handel-C, System C nebo System Verilog. Nástroj syntézy převede zapsaný algoritmus do architektury FPGA. Jelikož je zde velký stupeň volnosti, je vhodné některé informace o rozsahu paralelismu a časování doplnit jako omezení - o počtu násobiček či sčítaček, za kolik hodinových taktů se musí získat výsledek apod. Budoucí změny jsou potom jednoduché. Potřebujeme-li zvýšit výkon systému, změníme např. jen počet aritmetických jednotek a znovu provedeme syntézu. Nástroje syntézy na této algoritmické úrovni však dosud nejsou natolik kvalitní, aby se vyrovnali práci zkušeného návrháře, a používají se především u velkých obvodů pro číslicové zpracování signálu.

V jednom návrhu je samozřejmě možné používat různé kombinace zmíněných tří přístupů. Pro větší projekty je výhodné používat specializované nástroje pro návrh, které umožňují grafický i textový pohled na systém. Časté je např. používání grafického blokového schématu na nejvyšší hierarchické úrovni (**top level**) nebo návrh stavových automatů kreslením jejich diagramu přechodů. Grafické vstupy nebo abstraktnější textové popisy jsou pro člověka (návrháře) většinou srozumitelnější, a proto je výhodnější je používat (pokud nejsou na překážku efektivnosti popisu).

3.3 Verifikace

Verifikace (verification) představuje další významný krok v procesu návrhu obvodu. V rámci verifikačního procesu ověřujeme prostřednictvím simulací funkční správnost navrhovaného obvodu a dodržení časových parametrů. Na základě požadavků ve specifikaci systému je třeba sestavit seznam simulací, které budeme provádět, a pro každou simulaci navrhnout stimuly (vstupní testovací vektory), jež vybudí odpovídající chování obvodu. K tomu musíme specifikovat i očekávané odezvy obvodu, abychom mohli zkontrolovat správnost funkce. K zápisu vektorů používáme vhodné verifikační prostředí, např. testbench napsaný v jazyce HDL. Verifikační prostředí může i automaticky kontrolovat odezvy obvodu, což je u velkých obvodů v podstatě nutností. Simulaci je optimální provádět jak na RTL úrovni, tak na úrovni hradel. V procesu návrhu rozeznáváme následující simulace [42]:

- *Simulace RTL kódu* (RTL simulation) - simulujeme návrh na úrovni abstrakce RTL. Prováděná simulace nerespektuje reálná zpoždění na jednotlivých logických prvcích, ale je velmi rychlá. Je vhodná pro ověření správné funkce popisu RTL v jazyce HDL.
- *Simulace na úrovni hradel po syntéze* (post-synthesis netlist simulation) – simulujeme netlist po syntéze před rozmístěním a propojením logických prvků. Tato simulace je zejména u FPGA obvodů dosti nepřesná, protože zpoždění v propojovacích sítích je výrazně větší než ve vlastní logice. Proto se tato simulace

téměř nepoužívá. V podstatě je tato simulace vhodná jen pro ověření správné funkce syntézního nástroje.

- *Simulace na úrovni hradel s reálnými zpožděními* (back-annotated gate level simulation) – simulujeme netlist obvodu po rozmístění a propojení, tedy finální implementaci ve zvoleném FPGA obvodu. Simulátor potřebuje ke své činnosti informace o reálném zpoždění na jednotlivých spojích a prvcích. Dalším nezbytným vstupem simulátoru jsou modely obvodových prvků (vychází se z technologických knihoven dodaných výrobcem FPGA obvodu). Simulátory nejen modelují reálné funkční chování bloků na FPGA a zpoždění šíření signálů, ale kontrolují i dodržování správných časových parametrů (např. dodržení předstihu a přesahu u klopných obvodů).

Podcenění podrobné simulace v návrhovém procesu má často za následek chybně navržené systémy. Chyby v návrhu mohou vzniknout i vlivem chyb v návrhových nástrojích. Proto je simulaci věnovat dostatek času a návrhář by měl pokrývat simulacemi co nejvíce požadavků specifikace systému, simulace provádět jak na úrovni RTL, tak na úrovni hradel. U velkých projektů bývá čas na verifikaci 2krát až 3krát větší než čas strávený samotným návrhem. U menších projektů je někdy možné určité části návrhu nesimulovat a otestovat je až ve finální aplikaci. Obecně ale platí, že čím dříve v procesu návrhu chybu objevíme, tím nižší budou náklady na její odstranění.

Volba vhodných stimulů ani dokonalé provedení verifikace ale nezaručuje správnou funkci navrhovaného obvodu. Číslicový IO má obrovský počet stavů a reálně nelze simulacemi pokrýt všechny kombinace funkčních režimů, konfigurace obvodu či jeho časování. Proto je pro správnou funkci obvodu třeba zachovávat při návrhu určitá pravidla, o kterých se zmíníme zejména v kap. 4.5.

3.4 Implementace

Implementací označujeme proces, jehož výsledkem je vytvoření konfiguračních dat pro konkrétní FPGA obvod. Skládá se v podstatě ze tří hlavních kroků:

- *Syntéza* (synthesis) – obecně se jedná o proces konverze popisu systému z vyšší úrovně abstrakce na nižší. Dnes je nejběžnější konverze z kódu RTL (jeho syntetizovatelné podmnožiny) do seznamu logických prvků a jejich vzájemného propojení (netlist). Přitom jsou ve zdrojovém kódu detekovány registry a kombinační logika, jsou zakódovány stavy stavových automatů a rozpoznány speciální struktury (paměti, násobičky, aj.). Dále dochází k odstranění nepoužité logiky a dalším optimalizacím návrhu. Na syntézu mají vliv kromě HDL kódu také technologické knihovny a další omezení (tzv. constraints). Technologické knihovny obsahují popisy funkcí, časování a další důležité parametry všech prvků

dostupných ve zvoleném typu obvodu FPGA. Constraints dodává návrhář obvodu a jsou v nich definovány různé klíčové parametry, např. očekávanou maximální hodinou frekvenci nebo přiřazení vstupů a výstupů návrhu na piny obvodu FPGA.

- *Mapování na technologii* (mapping, u PLD se označovalo jako *fitting*) – obecně se jedná konverzi technologicky nezávislého netlistu do struktury dané technologie. U obvodů FPGA často provádí mapování už vlastní syntézní nástroj. Součástí mapování je i provádění další jednodušší optimalizace obvodu, seskupování LUTů a registrů do bloků apod.
- *Rozmístění a propojení (place and route)* – po namapování schématu je třeba provést v prvním kroku rozmístění logických prvků do matice FPGA obvodu. Pak jsou jejich pozice zafixovány a ve druhém kroku je prováděno propojování. Jsou identifikovány vhodné propojovací struktury (kanály, různé typy spojů, přepínací prvky) a postupně jsou zafixovány jednotlivé spoje mezi prvky. Snahou je optimalizovat propojení tak, aby zpoždění ve vodičích bylo minimální. Závisí to však jak na využití logiky v FPGA obvodu, tak na zaplnění propojovacích kanálů.

U velkých FPGA obvodů se můžeme setkat s tzv. *fyzickou syntézou* (physical synthesis), kdy celý proces implementace (syntéza, mapování, rozmístění a propojení) probíhá v několika cyklech. Po prvním cyklu jsou analyzovány kritické cesty zpoždění (jak v logice, tak v propojeních) a opětovně jsou syntetizovány příliš pomalé části obvodu. Celý proces je opakován, dokud není dosaženo požadovaných parametrů nebo do okamžiku, kdy je situace vyhodnocena jako bezvýchodná.

3.5 Konfigurace

Poslední fází návrhového procesu je naprogramování FPGA obvodu vytvořeným datovým souborem. Tato část závisí na principu programování obvodu (viz kap. 2.4). U FPGA na bázi antipojistek musíme ke konfiguraci použít externí programátor. Jelikož je tato technologie předurčena zejména pro hromadnou výrobu (větší série), umožňují některé programátory konfigurovat i více obvodů najednou. U obvodů založených na technologii EEPROM/flash se většinou programuje přímo v systému (ISP), na desce plošného spoje. Nejčastěji obvody podporují pro toto programování rozhraní standardu JTAG (IEEE Std. 1149.1-2001: IEEE Standard Test Access Port and Boundary-Scan Architecture), případně novější standard IEEE Std. 1532-2002: IEEE Standard for In-System Configuration of Programmable Devices). Propojení s počítačem, ve kterém je výhodné mít současně i návrhový systém, zajišťuje speciální kabel určený pro daný FPGA obvod. K programování je možné použít i externí programátor.

U nejpoužívanějších obvodů na bázi SRAM jsou možnosti konfigurace nejrozmanitější, z principu je ale samozřejmě nutné konfigurovat tyto obvody přímo na cílové desce systému.

Konfigurační soubor do nich lze nahrát jak přímo z počítače (přes vhodný programovací kabel rozhraním IEEE 1149.1 nebo IEEE 1532) nebo se může nahrát tato konfigurace z non-volatilní paměti (většinou externí flash paměť), což je v praktických aplikacích nejčastější. Nahrávání konfigurace z paměti nastává ihned po připojení napájecího napětí a doba konfigurace se pohybuje v řádu jednotek až stovek milisekund – podle typu připojení flash paměti a velikosti FPGA obvodu (při použití JTAG rozhraní může nahrání trvat i několik sekund). Paměti se připojují k FPGA buď přes jednoduché sériové rozhraní (nejčastěji SPI, Serial Peripheral Interface) anebo paralelně, což zkracuje dobu konfigurace. Paralelní mód má standardně šířku datové sběrnice 8 bitů, ale u nejnovějších obvodů může mít 16 nebo 32 bitů. Vlastní přenos dat z konfigurační paměti do FPGA obvodu si nejčastěji řídí FPGA obvod sám (generuje hodinové a řídicí signály pro paměť), mluvíme o tzv. *Master* módu. Aktivní při nahrávání konfigurace však může být např. externí mikrokontrolér, pak je FPGA obvod v tzv. módu *Slave*. Do konfigurační paměti je možné uložit konfigurační data více programovatelných obvodů, a tak z jedné paměti konfigurovat několik obvodů, a to dokonce i různých typů a velikostí (ale od jednoho výrobce). V tomto případě jsou konfigurované obvody zapojeny v řetězci, první FPGA obvod je nastaven do konfiguračního módu Master a ostatní jsou Slave (hodinová frekvence konfiguračních dat je generována prvním obvodem v řetězci). Každý z různých režimů konfigurace má své výhody i nevýhody – záleží na době konfigurace, na počtu nevyužitých vývodů obvodu FPGA a na prostoru na desce plošných spojů pro konfigurační paměť.

Dnešní FPGA obvody jsou značně rozsáhlé, a proto i jejich konfigurační soubory mají velkou velikost (řádově desítky Mb, největší obvody až 170 Mb – Altera Stratix IV). Velikost konfiguračního souboru odpovídá počtu všech konfigurovaných paměťových buněk v daném FPGA – vždy se konfigurují všechny buňky bez ohledu na velikost konkrétního obvodového návrhu. Velikost konfiguračního souboru je možné zmenšit kompresí dat, kterou současné FPGA podporují. Díky ní je možné zmenšit velikost konfiguračních dat o 30-50 % (návrhový systém bez problémů vytvoří zkomprimovaný bitstream buď ve fázi implementace, nebo jej můžeme dodatečně zkomprimovat). Ušetříme tak jednak prostor ve flash paměti a jednak konfigurace FPGA obvodu probíhá rychleji. Dekomprese se provádí on-line během procesu konfigurace. Velikosti flash pamětí pro uložení konfigurace bývají podle velikosti a počtu připojených obvodů od 1 Mbitu do 128 Mbitů.

Tradiční postup návrhu (*design flow*) FPGA obvodu, tak jak jsme jej nyní popsali, tj. specifikace funkce, vstup návrhu, verifikace a implementace, se často nazývá „*vodopád*“ (waterfall). V tomto modelu návrh postupuje po krocích, z jedné fáze do druhé a již se nevrací zpět. Pro návrhy složitějších systémů, zvláště pokud na projektu pracuje více návrhářů, se však jednoznačně jeví vhodnější model „*spirála*“ (spiral), kdy se většina fází opakuje několikrát s postupně se upřesňujícím a zlepšujícím řešením [43].

4 Systémový návrh

Systémový návrh je úvodní a z hlediska dosažení optimálního výsledku asi i nejvýznamnější částí návrhu nejen FPGA obvodů. Jeho cílem je návrh celého systému zejména z hlediska jeho funkce, architektury jednotlivých bloků včetně optimalizace návrhu z různých hledisek a příp. ověření funkce vhodně zvolenými testovacími signály. Obecně probíhá jako víceúrovňový hierarchický iterační proces na různých úrovních abstrakce. Stěžejní úlohou systémového návrhu je nalezení obvodové struktury vyhovující zadaným požadavkům. Jde ale o úlohu mnohoznačnou. Mnohoznačnost vyplývá jak z volby architektury, tak z konkrétního řešení dílčích funkčních bloků v hierarchii celého systému.

Systémový návrh je tvůrčí inženýrská práce, ve které zatím rozhodující roli hraje intuitivní myšlení návrháře, tj. souhrn jeho znalostí, technických schopností a zkušeností. Intuitivní přístup k návrhu zpravidla využívá heuristické principy, které ve většině případů vedou k cíli, (tj. k nalezení vyhovujícího řešení), ale nezaručují optimálnost řešení vzhledem k zadaným požadavkům. Jedním z cílů této kapitoly je shrnout pravidla systémového návrhu, kterými lze dosáhnout spolehlivě fungujícího řešení.

Stanovit nějaký univerzální směr postupu v systémovém návrhu je velice obtížné. Obecně se rozlišují dva základní modely přístupu k řešení návrhu [44]:

- postup *zdola-nahoru* (bottom-up) - nejprve se vytvoří dílčí bloky (případně jsou již připravené) a ty se pak sestavují do větších a větších celků,
- postup *shora-dolů* (top-down) – nejprve se definuje funkce navrhovaného systému jako celku; pak se postupně vyčleňují bloky, jejichž funkce se specifikuje spolu s vzájemnou návazností jednotlivých bloků.

Přístup **zdola-nahoru** byl typický při sestavování větších číslicových systémů z diskrétních integrovaných obvodů nižší úrovně. Tato metoda je obecně méně náročná na zkušenosti návrháře, ale na druhou stranu ne vždy vede k dosažení optimálních výsledků. U rozsáhlých systémů je mnohem perspektivnější metoda **shora-dolů**, která je vesměs podporována v moderních návrhových systémech. Popis na abstraktnější úrovni je možné prostředky syntézy převádět až na nejnižší úroveň. Přístupy zdola-nahoru a shora-dolů může být účelné v různých fázích návrhu vzájemně kombinovat.

Při specifikaci funkce systému je třeba zvážit, zda řešená úloha je pro implementaci do FPGA vhodná. Návrhář má při řešení svého problému na nejvyšších úrovních abstrakce obecně možnost volby mezi softwarovým a hardwarovým řešením. Zejména podle požadovaných dynamických vlastností, rozsáhlosti a ceny výsledného řešení musí volit mezi mnoha možnostmi – od univerzálního procesoru, přes signálový procesor, programovatelné hradlové pole až po speciální ASIC obvod. Přesná hranice mezi jednotlivými druhy řešení

jednak neexistuje a jednak se s vývojem integrovaných systémů neustále mění. FPGA obvody budeme preferovat tam, kde se vyžaduje zpracování kritických částí algoritmů, ve kterých by převážně sekvenční procesorové řešení bylo pomalé. Rychlost hardwarové implementace je většinou omezená pouze vlastní paralelizovatelností řešeného problému.

U čistě procesorových řešení je v podstatě předem dána architektura a návrhář může vybírat jen z řad procesorů s různou výkonností. Výsledné řešení má horší dynamické vlastnosti, je však zpravidla levnější a vývoj je relativně rychlý. Jak již bylo naznačeno v kap. 2.2.4, není v dnešní době problém mít i v obvodu FPGA procesorové jádro, a to buď hardwarové, nebo softwarové. Tyto systémy mají výrazně vyšší flexibilitu a podporují souběžný a kooperující návrh hardwaru a softwaru. Z tohoto důvodu se tento návrh často označuje jako **HW/SW co-design**. Jelikož se systémový návrh tvoří jako celek, problémy ve spolupráci hardwarové a softwarové části se vesměs odhalí v počátku, a tím se zrychluje celý vývoj. Při tradičním odděleném návrhu HW a SW většinou až na konci vývoje návrháři zjišťovali vzájemné „nekompatibility“, které prodlužovaly čas vývoje, resp. čas uvedení výrobku na trh.

4.1 Makrobloky

V předchozím textu jsme se zmiňovali o perspektivnosti směru návrhu shora-dolů, zvláště u rozsáhlejších systémů. Tento postup je možné jistě doporučit při návrhu zcela nových částí. S rostoucí složitostí čipů (a tím i rozsáhlosti návrhu) není možné z hlediska doby návrhu začínat vždy od počátku a je nutné využít některé bloky již dříve navržené. Využívání hotových částí tedy částečně vede ke směru návrhu zdola-nahoru, nebo spíše ke kombinaci obou postupů, jak již bylo naznačeno v předchozích odstavcích.

Samostatné, znovu použitelné komponenty bývají označovány jako **makrobloky**, případně i jinými termíny – IP (Intellectual Property, duševní vlastnictví) makra, IP jádra, jádra, anglicky nejčastěji cores, IP cores, reusable cores, atp. Jedná se vlastně o předem navržené bloky, které je možné zaintegrovat do logického systému, a velmi výrazně tak zkrátit dobu návrhu. Jádrem může být např. aritmeticko-logická jednotka, sériové rozhraní, rozhraní PCI nebo i celý procesor. Vzniklo i mnoho firem, které navrhují pouze IP makra a úspěšně s nimi obchodují (tzv. *IP provider*). Vzhledem k časové náročnosti se totiž nevyplatí si určité makro vyvíjet vlastními prostředky a je výhodnější jej raději koupit včetně veškeré dokumentace, příp. simulačním modelem. Na trhu jsou dnes k dispozici ekvivalenty většiny samostatně používaných procesorů, součástek a rozhraní.

Makra se nejčastěji rozdělují podle toho, v jaké formě se vyskytují, což v podstatě určuje okamžik, kdy vstupují do návrhového procesu:

- *Softwarová makra* – jsou ve formě popisu v HDL jazyku; popis je obvykle technologicky nezávislý a teoreticky může být syntetizován do libovolné

technologie. Tato makra umožňují velkou míru flexibility, ale musí projít celým návrhovým procesem. Konkrétní dynamické parametry makra budou pak záviset na vlastní implementaci.

- *Firmwarová makra* – je dodáván „netlist“ v dané technologii; vhodné právě pro FPGA obvody. Protože kvalita rozmístění a propojení výrazně ovlivňuje rychlost obvodu, dodávají se někdy s netlistem pomocné soubory s direktivami pro rozmísťovací a propojovací program.
- *Hardwarová makra* – je poskytována hotová topografie čipu (layout), tedy úzce souvisí s cílovou technologií; vhodné pro ASIC obvody. Návrhový proces zachází s těmito makry jako s černými skříňkami a je nemožné v nich cokoli měnit.

Očekává se, že využití maker je jedinou možností, jak v budoucnu v rozumném čase navrhnout složité systémy jak v obvodech ASIC, tak v FPGA. Snahou je, aby byla navržená makra co nejvíce přenositelná, a tedy standardizovaná – jak po stránce technické, tak dokumentační.

S potřebou usnadnění složitějších návrhů, s přenositelností navržených celků na jinou platformu či technologii a s jejich standardizací souvisí **knihovny parametrizovatelných modulů** (tzv. LPM, Library of Parameterized Modules). Jedná se o parametrizovatelné funkce (funkční bloky) jako např. hradla, multiplexory, klopné obvody, čítače, stavové automaty, aritmetické a paměťové funkce. Většina funkcí je parametrizovatelná i několika parametry (bitová šířka, zřetězení, směr čítání, předvolba, zobrazení dat apod.), což umožňuje z jedné funkce vytvořit mnoho modifikovaných variant. Tyto funkce jsou podporovány jak výrobci návrhových nástrojů, tak výrobci FPGA obvodů. Standard LPM je součástí standardu EDIF (Electronic Design Interface Format), což je formát pro přenos návrhu mezi návrhovými nástroji různých výrobců a popisuje syntaxi reprezentující logické operace netlistu. Knihovny LPM jsou volně šiřitelné, což také přispívá k jejich využívání.

4.2 Softwarové procesory

Navažme na předchozí kapitolu o IP jádrech a zmiňme se o stále častěji používaných procesorových jádrech v obvodech FPGA. Nacházejí uplatnění zejména při realizaci rozsáhlejších algoritmů nebo aritmetických operací, jejichž čistě hardwarová implementace by byla neúměrně rozsáhlá, a tím i drahá. Výhodou může být i rychlejší návrh algoritmu a jeho pozdější snadné úpravy. I když se v této práci zaměřujeme převážně na hardwarovou realizaci systémů v FPGA, nemůžeme procesorová řešení (alespoň částí navrhovaných systémů) zcela pominout – zejména v souvislosti s již zmiňovaným trendem HW/SW co-designu. V kap. 2.2.4 jsme se již stručně dotknuli hardwarových procesorových jader, která momentálně ustupují softwarovým procesorům. Všichni významnější výrobci FPGA obvodů nabízejí nějaké varianty softwarových procesorů, resp. (mikro)řadičů, např.:

- Xilinx – MicroBlaze (32bitový), PicoBlaze (8bitový),
- Altera – Nios II (32bitový),
- Lattice – LatticeMico32 (32bitový), LatticeMico8 (8bitový).

Každý z uvedených procesorů má několik variant podle výkonnosti či podle typu cílového obvodu daného výrobce. Lze samozřejmě implementovat i univerzální procesory (IP jádra) jiných dodavatelů, které mohou například nahrazovat clony používaných diskrétních procesorů (Intel 8051, ARM Cortex-M1, Freescale V1 ColdFire aj.).

Softwarové procesory jsou převážně založeny na harvardské architektuře (mají oddělenou paměť pro program a paměť pro data), jsou typu RISC a z hlediska architektury souboru instrukcí jsou registrově orientované. Jednodušší varianty jsou jednoduché subskalární procesory, složitější pak skalární procesory s několikastupňovou pipeline. Sběrnice jsou buď dány konkrétním výrobcem, nebo si je návrhář navrhne sám. K obecným výhodám softwarových procesorů zejména patří:

- velká flexibilita – architektura lze přizpůsobit řešenému problému (konfigurovat lze jak vlastní procesorové jádro, tak jeho periferie či stykové obvody),
- do jednoho FPGA lze podle potřeby implementovat i více CPU jader,
- do instrukčního souboru lze doplňovat další potřebné instrukce,
- snadná realizace a připojení periferií,
- při vývoji nejsou třeba kromě FPGA obvodu žádné další součástky, v případě potřeby lze snadno a rychle přejít k jinému typu procesoru či periferií,
- snadný pozdější upgrade jak softwaru, tak hardwarové části.

Softwarové procesory jsou dodávány v syntetizovatelné podobě (často v některém z HDL jazyků), jednodušší verze jsou většinou volně ke stažení (freeware), k výkonnějším variantám je třeba zakoupit licenci. Konfigurování SW procesorů, včetně jejich periferií, probíhá většinou ve specializovaných vývojových prostředích konkrétních výrobců, nejznámější jsou EDK (Embedded Development Kit) firmy Xilinx nebo SOPC (System On a Programmable Chip) Buider firmy Altera. Tato prostředí umožňují integrovat do výsledného systému jak standardní makrobloky, tak vlastní komponenty či IP jádra třetích výrobců. Pro vývoj aplikací jsou opět k dispozici nástroje výrobců FPGA nebo volné GNU nástroje, převážně podporující jazyky C a C++. Před použitím konkrétního jádra bychom se měli seznámit s jeho podporou v softwarových prostředcích (v kompilátorech či simulátorech) a rozhodně nevyhledávat neověřené SW procesory. Více o implementaci SW procesorů lze nalézt např. v [16].

Závěrem si ještě pro představu naznačme, kolik logických buněk potřebují procesorová jádra ke své činnosti a jakou mají výkonnost. Obě veličiny samozřejmě závisí na druhu použitého SW procesoru a také na cílovém FPGA obvodu. Jednoduchý PicoBlaze zabere v obvodu Xilinx Spartan 3E cca 200 logických buněk, složitější MicroBlaze zhruba 1600 buněk. Firma Altera uvádí pro svůj procesor Nios II zaplnění 600-700 LE (varianta

Economy), 1200-1400 LE (varianta Standard) a 1400-1800 LE (varianta Fast). K tomu je třeba přidat část blokové RAM (podle velikosti paměťového prostoru) a případně další logické buňky na přídavné periferie a stykové obvody. Výkonnost SW procesorů se pohybuje se zhruba od 0,1 do 1,2 DMIPS/MHz. Při taktovacích frekvencích v řádech desítek až stovek MHz (max. do 300 MHz) jsou absolutní výkonnosti aktuálně do 340 DMIPS [45], což je méně než u procesorů hardwarových.

4.3 Volba algoritmizace úloh

Volba algoritmu řešení konkrétní funkce (syntéza) je obecně mnohoznačná úloha. Kvalita výsledného digitálního návrhu se v podstatě posuzuje podle tří fyzikálních charakteristik – **maximální rychlosti** (maximální možné hodinové frekvence), **množství použité logiky** (resp. zabrané plochy na čipu) a **výkonové spotřeby** (ztrátovým teplem vznikajícím na součástce). Tyto charakteristiky jsou vzájemně protichůdné a obtížně se hledá vzájemný kompromis.

Systémový inženýr musí znát prioritu jednotlivých charakteristik a svým návrhem je aktivně ovlivňovat. Nejvíce může návrh ovlivnit na vyšších úrovních abstrakce, zejména při výběru algoritmizace řešení. Na nižších úrovních pak výsledek hodně závisí na návrhovém systému, na jeho prostředcích syntézy. I když se zdá tato nižší úroveň pro návrháře transparentní, lze ji významně ovlivňovat vhodným nastavením různých parametrů syntezátoru. Proto se v dalších podkapitolách o některých významných nastaveních zmíníme.

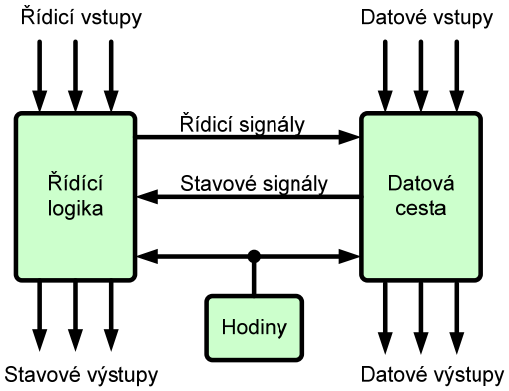
Jedním z prvních problémů, které musí návrhář řešit na nejvyšší úrovni (top level), je rozdělení systému do dílčích bloků (komponent). Bývá dobrým zvykem na nejvyšší úrovni hierarchie rozložit systém „čistě“ na funkční bloky (komponenty) a neumisťovat zde žádnou, byť jednoduchou logiku (tzv. „glue logic“).

Problém rozčlenění na podsystémy může souviset i s rozdělením na část řešenou procesorem (jak externí součástkou, tak třeba softwarovým CPU) a část řešenou logikou v hardwaru (tzv. *HW-SW partitioning*), nebo s rozdělením rozsáhlého návrhu do více FPGA obvodů. Zde bychom měli respektovat tzv. *princip lokality*, který „říká“, že části, které spolu nejvíce komunikují, by měly být nejblíže. Tento princip platí jak na nejvyšších úrovních abstrakce, které může návrhář asi nejvíce ovlivnit, tak na obvodové úrovni, která je především řešena návrhovým systémem. Správné rozdělení do bloků na všech úrovních abstrakce urychluje běh syntézy a lze dosáhnout lepších výsledných parametrů.

Velikost logické funkce uvnitř jednoho FPGA obvodu by měla být v rovnováze k možnostem jeho vstupních a výstupních pinů. Nemá smysl umísťovat do FPGA více logických funkcí, pokud k nim, resp. od nich, nejsme schopni přes I/O piny přivádět a odvádět zpracovávaná data. V některých případech je možné některé piny *multiplexovat*, což ale nezvýší propustnost vstupů/výstupů. Jinou možností je přivádět/odvádět data po *rychlých*

sériových sběrnicích. V moderních FPGA obvodech jsou totiž I/O buňky schopné komunikovat na vyšší frekvenci, než jakou zvládá vnitřní logika. Struktura I/O buňky obsahuje rychlé *deserializéry* (serial-to-parallel converter) a *serializéry* (parallel-to-serial converter), které mají dokonce programovatelnou šířku slova (nejčastěji 2 až 10 bitů). Vstupní sériový tok dat převedeme tedy na paralelní slova, která po zpracování uvnitř FPGA (na nižší pracovní frekvenci) opět konvertujeme na rychlou sériovou sběrnici.

Představme si, že jsme na vyšších úrovních hierarchie rozčlenili systém na jednotlivé subsystémy a nyní je třeba vytvořit popis na meziregistrové úrovni RTL (nejčastěji v některém z HDL jazyků). Charakteristiku této úrovně jsme již popisovali v kap. 3.2. Zde bych upozornil na jednu významnou zásadu, jejíž respektování výrazně přispívá k návrhu spolehlivého a v budoucnu snadno modifikovatelného systému. Touto zásadou je důsledné rozdělení navrhovaného systému na *část vytvářející řídicí signály* a na *část zpracovávající data* (obr. 25), a pro jednotlivé bloky specifikovat jejich rozhraní, funkce i hodinové signály. Každou ze dvou uvedených částí je možné případně dále rozdělit do menších a relativně samostatných bloků podle implementovaných funkcí.



Obr. 25: Členění návrhu na datovou a řídicí část

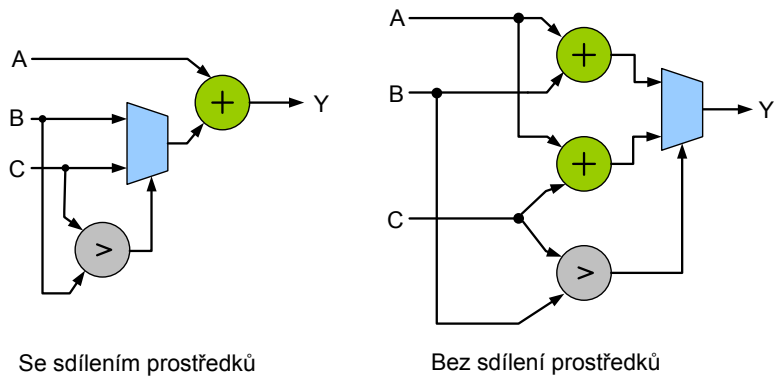
Řídicí část (řadič) je možné řešit v principu dvěma implementacemi – buď **pamětí s mikroprogramem**, nebo pevně definovaným **stavovým automatem**. Programovatelné řadiče jsou univerzálnější a snadněji měnitelné, ale jsou náročné na plochu paměťové matice zejména u větších řešení. V FPGA se implementují většinou do blokových pamětí. Stavové automaty spotřebují méně paměťových buněk, ale jsou zpravidla náročnější kombinační logiku. Realizaci stavových automatů si rozebereme v kap. 4.3.2. V následující podkapitole se věnujme rozboru návrhu **datové části (cesty)**, která je základem kvality celého řešení systému. Datová část se obecně skládá z výpočetních jednotek řešících vlastní operace (ALU, MAC, aj.), datových registrů pro uchování jednotlivých proměnných (pamětí, zásobníků, aj.) a komunikačních datových sběrnic.

4.3.1 Návrh datové části

Předpokládejme, že jsme celý systém vhodně rozdělili na dílčí komponenty s definovanými funkcemi, a nyní přistoupíme k volbě algoritmizace při zpracování dat. Algoritmy lze v principu řešit dvěma krajními implementacemi – *sériovou* a *paralelní*. Každá varianta má jistě své přednosti i nedostatky. Jejich hlavní rozdíly (zejména z hlediska šířky zpracovávaných slov a latence zpracování) lze podle [46] shrnout do následujících bodů:

- paralelní struktury mají díky paralelnímu zpracování dat vesměs větší propustnost, nárůst nemusí být přímo úměrný délce slova,
- paralelní struktury zabírají výrazně větší plochu, zejména při větší velikosti datového slova; nárůst plochy může být s délkou slova lineární, ale i vyšší (např. kvadratický),
- sériové struktury mají zpravidla jednodušší kombinační logiku ve srovnání s paralelní implementací, a proto je lze taktovat vyšší pracovní frekvencí,
- sériové implementace nejsou závislé na šířce datového slova, a proto je lze použít u systémů, kde délka slova není předem známa nebo se může i dynamicky měnit během práce obvodu,
- sériové obvody zpracovávají data bit po bitu, a proto musí obsahovat i paměť pro uchování výsledků operací s předchozím bitem.

Z uvedených vlastností je zřejmé, že nelze obecně preferovat sériové či paralelní algoritmy. Maximální paralelnost je v hardwarových strukturách v principu omezena jen vlastní paralelizovatelností problému, ale za cenu nárůstu potřebné logiky. V některých případech může ale i sériový algoritmus dosáhnout vyšší efektivity. V počátcích programovatelných zakázkových obvodů bylo běžnější sériové zpracování dat, především díky malé velikosti a vysoké ceně zmiňovaných obvodů. S rostoucí velikostí hradlových polí a s poklesem ceny za logický prvek přecházejí sériové architektury pozvolna na paralelní. V mnoha případech se zdá být pro číslicové zpracování signálu v FPGA obvodech optimální kompromis v podobě sériově-paralelní implementace. Míru paralelizace může systémový návrhář ovlivňovat na různých úrovních abstrakce návrhu – stupeň paralelizace lze volit jak na úrovni architektury, tak na úrovni řešení jednotlivých obvodů.

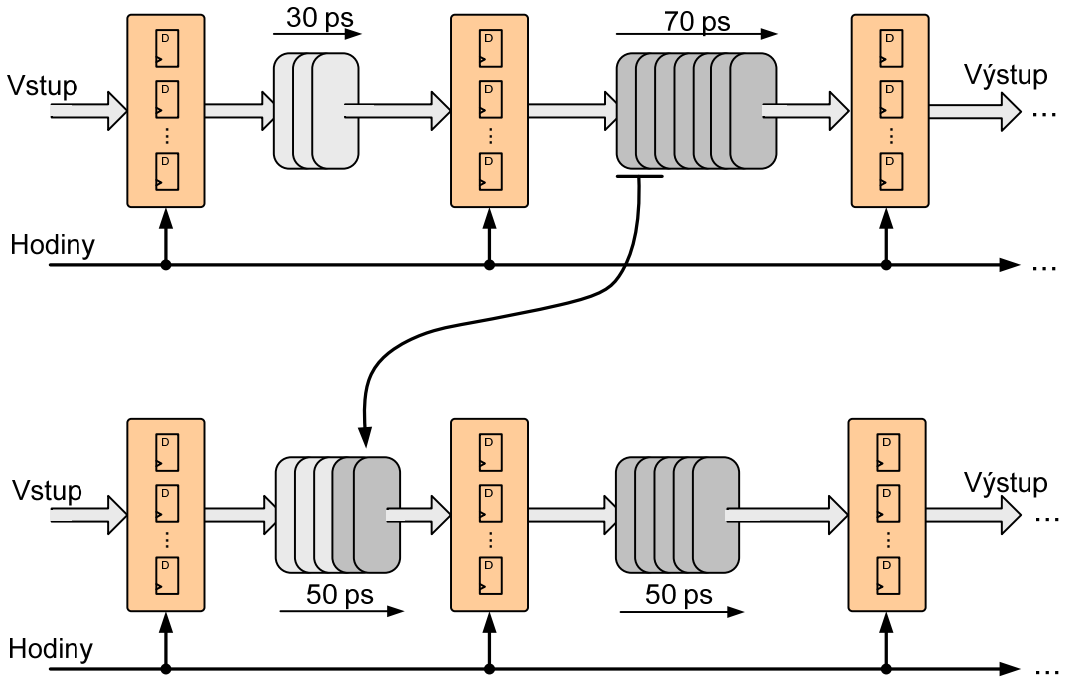


Obr. 26: Přiklad techniky sdílení prostředků

Drobnější úpravy v paralelnosti či sériovosti algoritmu na úrovni meziregistrových přenosů RTL může provádět sám návrhový systém (nástroj syntézy). Velký vliv na výsledek syntézy má jak optimální nastavení parametrů návrhového systému, tak vhodný zápis funkce v jazyce HDL. Jako významnou techniku v této oblasti bych označil **sdílení prostředků** (*resource sharing*), někdy uváděné také jako *časový multiplex* (TDM, Time Division Multiplex). Principem této techniky je opakované používání vybraných funkčních bloků při řešení daného algoritmu. Aplikací této techniky typicky na vyšších úrovních abstrakce řešíme kompromis mezi rychlostí výsledného algoritmu a plochou (logickými obvody), kterou návrh zaplní. Pro ukázku byl vybrán příklad z [47], kde je řešen běžný zápis v jazyce VHDL:

```
if (B > C) then Y = A + B; else Y = A + C; end if;
```

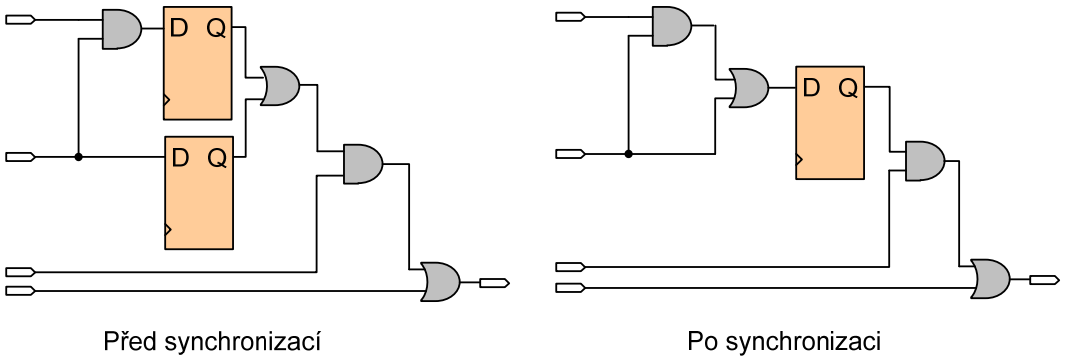
Příklad byl syntetizován v návrhovém systému Quartus II do FPGA řady Cyclone III (EP3C5F256A7), operandy *A*, *B*, *C* i výsledek *Y* byly 32bitové. Pokud bylo zapnuté sdílení prostředků, výsledkem syntézy bylo levé zapojení na obr. 26. Návrh zaplnil 64 LE a nejdelší kritická cesta byla 28,5 ns (tj. max. frekvence 35,1 MHz). Pokud se sdílení prostředků nepoužilo (implicitní nastavení), výsledek vedl na zapojení na obr. 26 vpravo. Návrh tentokrát zaplnil 96 LE (je zde navíc jedna 32bitová sčítačka), ale kritická cesta má jen 20,3 ns, tj. lze taktovat 49,3 MHz. Porovnáním výsledků je patrné, že druhý případ potřebuje o 50 % více LE, ale může pracovat o 40 % rychleji. Vlastní zdrojový soubor v jazyce VHDL je v příloze práce. S dalšími příklady sdílení prostředků se můžeme setkat např. v kap. 5 při diskusi implementace aritmetických operátorů.



Obr. 27: Princip techniky synchronizace registrů

V předchozích odstavcích jsme se zmiňovali o volbě mezi sériovými nebo paralelními algoritmy. Obě tyto varianty lze obecně zrychlit použitím optimalizačních technik, z nichž nejčastěji se aplikují dvě – *synchronizace registrů* (tzv. *register retiming*) a *proudové zpracování* (tzv. *pipelining*). Obě vycházejí z rozložení logického návrhu na části čistě sekvenční (samotné klopné obvody či registry) a části kombinační (ostatní logické obvody). V principu jde tedy o rozdělení na „sendvičovou" strukturu naznačenou na obr. 27. Maximální pracovní kmitočet celého návrhu je v podstatě omezen nejdelší kritickou cestou, tj. největším zpožděním průchodu signálu kombinační logikou mezi dvěma registry. Tuto cestu lze zjistit v návrhovém systému provedením statické časové analýzy (STA, Static Timing Analysis) a není třeba provádět časové simulace.

Snahou tedy je, aby množství kombinační logiky mezi registry bylo vyrovnané. Pokud tomu tak není, lze v některých případech část kombinační logiky přesunout mezi registry tak, aby se zkrátila nejdelší kritická cesta. Tato technika se označuje jako **synchronizace registrů** (obr. 27) a bývá prováděna vesměs nástroji syntézy (většinou automaticky, je-li povolena). Na obr. 28 je pro představu uveden jednoduchý příklad synchronizace registrů [48].



Obr. 28: Příklad synchronizace registrů

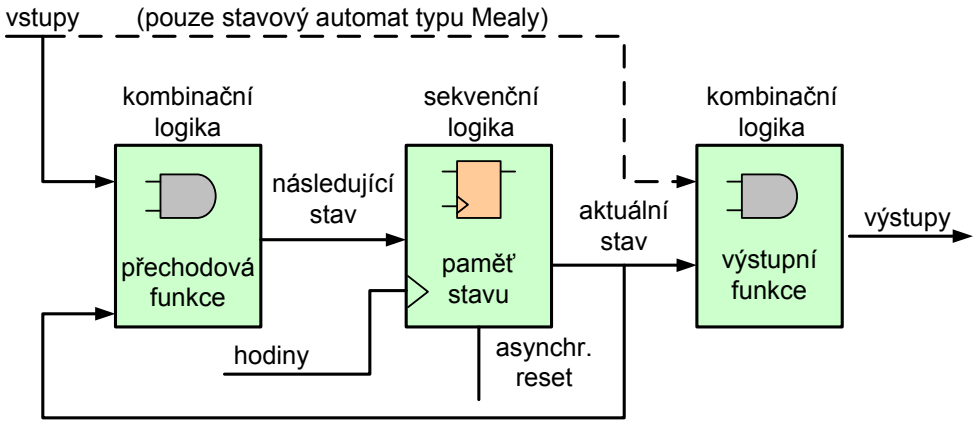
Další možností, jak zrychlit pracovní frekvenci, je vložení dalších registrů do dlouhých kritických cest (registry jsou taktovány hodinovou frekvencí sousedních registrů). Tato technika, označovaná jako **proudové zpracování** (zřetězení), kritickou cestu rozdělí na menší části, ale na druhou stranu vnese do cesty určitou latenci, která musí být akceptovatelná. Zároveň se do cesty přidá zpoždění registru, které by vzhledem ke zpoždění kombinační logiky mělo být zanedbatelné. V principu bychom mohli neustále rozdělovat kombinační logiku pomocí registrů, a tak neustále zrychlovat celý systém. Bohužel vložené zpoždění reprezentované registrem se tak stává ve srovnání se zpožděním kombinační logiky stále významnější, a tím méně efektivní. Vzrůstající počet registrů způsobuje kromě větší latence také větší zatížení hodinových rozvodů a nezanedbatelně vzrůstá i spotřeba celého systému.

4.3.2 Návrh řídicí části

Řadič na základě stavových informací řídí datovou cestu. Jak jsme se již dříve zmínili, lze řadič řešit mikroprogramem nebo stavovým automatem. V této kapitole je věnujme hardwarovým řadičům, které řešíme ve formě synchronního sekvenčního obvodu konečným stavovým automatem. Lze obecně použít i více vzájemně provázaných automatů, což ale může být zvláště při různých hodinových signálech nebezpečné z hlediska vzniku metastabilit (viz kap. 4.5.1). Konečný automat má konečný počet vstupních, výstupních a vnitřních stavů. Stavem rozumějme kombinaci hodnot konkrétních většinou dvouhodnotových signálů, na které můžeme pohlížet jako na binární vektor (slovo).

Pro praktické realizace číslicových systémů v podstatě používáme dva typy konečných automatů – *Moore* a *Mealy*. Oba typy je možné mezi sebou vzájemně převádět. Podle podmínky pro změnu vnitřního stavu rozlišujeme automaty asynchronní a synchronní. Asynchronní automat mění svůj vnitřní stav s malým zpožděním po změně vstupního stavu. Tento druh automatů je pro rozsáhlé číslicové systémy zcela nevhodný (viz kap. 4.5) a nepoužívá se. My budeme dále uvažovat pouze automaty synchronní, u kterých nastává změna stavu s příchodem hodinových impulsů a jejichž paměťové prvky jsou založeny na synchronních klopných obvodech (nejčastěji typu D). Tyto stavové automaty se obecně skládají ze tří částí [49] – viz obr. 29:

- *přechodové funkce*, která ze vstupního a z aktuálního vnitřního stavu připravuje příští vnitřní stav,
- *stavového registru*, který s příchodem hodinového signálu přepíše aktuální vnitřní stav (na svém výstupu) připraveným stavem na svém vstupu (tj. výstupem přechodové funkce),
- *výstupní funkce*, která připravuje nový výstupní stav automatu – vstupem je buď pouze aktuální vnitřní stav (automat typu Moore) nebo aktuální vnitřní stav a vstupní stav (automat typu Mealy).



Obr. 29: Blokové schéma stavového automatu

Přechodová a výstupní funkce jsou čistě kombinační logické funkce, stavový registr je jedinou sekvenční částí. Z uvedených blokových schémat je tedy zřejmé, že při požadavku na asynchronní reset je nutné jej implementovat do stavového registru, který jako jediný blok má paměťovou funkci. Naopak synchronní reset by měl být součástí přechodové funkce, která generuje nový (následující) vnitřní stav. Při aktivním synchronním resetu tedy vlastně přechodová funkce připraví na svém výstupu „počáteční“ stav, který se následně s aktivní hodinovou hranou přepíše do aktuálního stavu. Řešení synchronního resetu/setu ve stavovém registru může v podstatě v některých případech (pokud například klopné obvody nemají přímo zaintegrován tento reset/set) znamenat formální přesun části kombinační logiky z přechodové funkce do „registru“.

Dále je třeba si uvědomit, že výstupy stavového automatu jsou výstupy z kombinační logické funkce a mohou obecně vykazovat hazardy, proto je v některých případech vhodné tyto výstupy synchronizovat dalšími registry (i za cenu zvýšení latence).

Při popisu stavového automatu v některém z HDL jazyků se doporučuje důsledně respektovat rozdělení na části podle obr. 29 a každou zmiňovanou část popsat vhodnou formou samostatně. Zlepšíme tak čitelnost vlastního popisu a hlavně umožníme návrhovému systému rozpoznat popis stavového automatu, a tak zvolit optimální syntézu. Nerespektování tohoto rozdělení je častou chybou začátečníků, podobně jako nerozdělování systému na datovou a řídicí část.

Za určitých okolností je možné popis sekvenčního systému konečným stavovým automatem (složeného ze tří částí) nerespektovat. Příkladem mohou být speciální typy sekvenčních obvodů, na které je možné pohlížet jako na speciální případy stavových automatů a v jazycích HDL mají vesměs vlastní vhodnější konstrukce. Jsou to nejčastěji:

- *čítače* – stavový automat bez vstupního stavu a s jednotkovou výstupní funkcí (jednotkovou funkcí máme na mysli funkci bez implementace jakékoli logiky),
- *registry* – automat s jednotkou výstupní funkcí, přechodová funkce nezávisí na aktuálním vnitřním stavu a je také jednotková,
- *posuvné registry* – automat s jednotkovou výstupní funkcí, přechodová funkce je v podstatě také jednotková, ale respektuje posun bitů.

S rozvojem možností návrhových systémů a s přechodem popisu na stále abstraktnější formy je však možné se za určitých okolností od zmiňovaného popisu automatu třemi částmi rovněž oprostit (například použitím grafické formy zadání stavovým diagramem nebo použití abstraktního popisu ve vhodném programovacím jazyku vyšší úrovně).

Kódování vnitřních stavů

S návrhem stavových automatů úzce souvisí relativně významná fáze návrhu - volba kódování vnitřních stavů. Kódováním označujeme přiřazené konkrétní kombinace hodnot jednotlivých bitů každému stavu. Pro synchronní sekvenční obvody platí, že při jakémkoli kódování lze vždy navrhnout přechodovou a výstupní funkci tak, aby bylo dosaženo požadované funkce. Volba kódování má však zpravidla vliv na vlastnosti výsledného automatu, zejména jeho složitost, a následně na dynamické vlastnosti (maximální kmitočet hodinových impulzů). Potřebujeme-li zakódovat „n_s“ vnitřních stavů, potřebujeme k tomu minimálně „k“ bitů, přičemž platí:

$$n_s \leq 2^k \tag{4.1}$$

V některých případech je výhodné zavést větší než minimální počet bitů, tím však vzrůstá počet nevyužitých stavů, které mohou způsobit dodatečné komplikace. Pokud se automat dostane do nevyužitého stavu (například po zapnutí napájení nebo vlivem rušivých pulzů), nemusí se z něj již za normálních podmínek dostat.

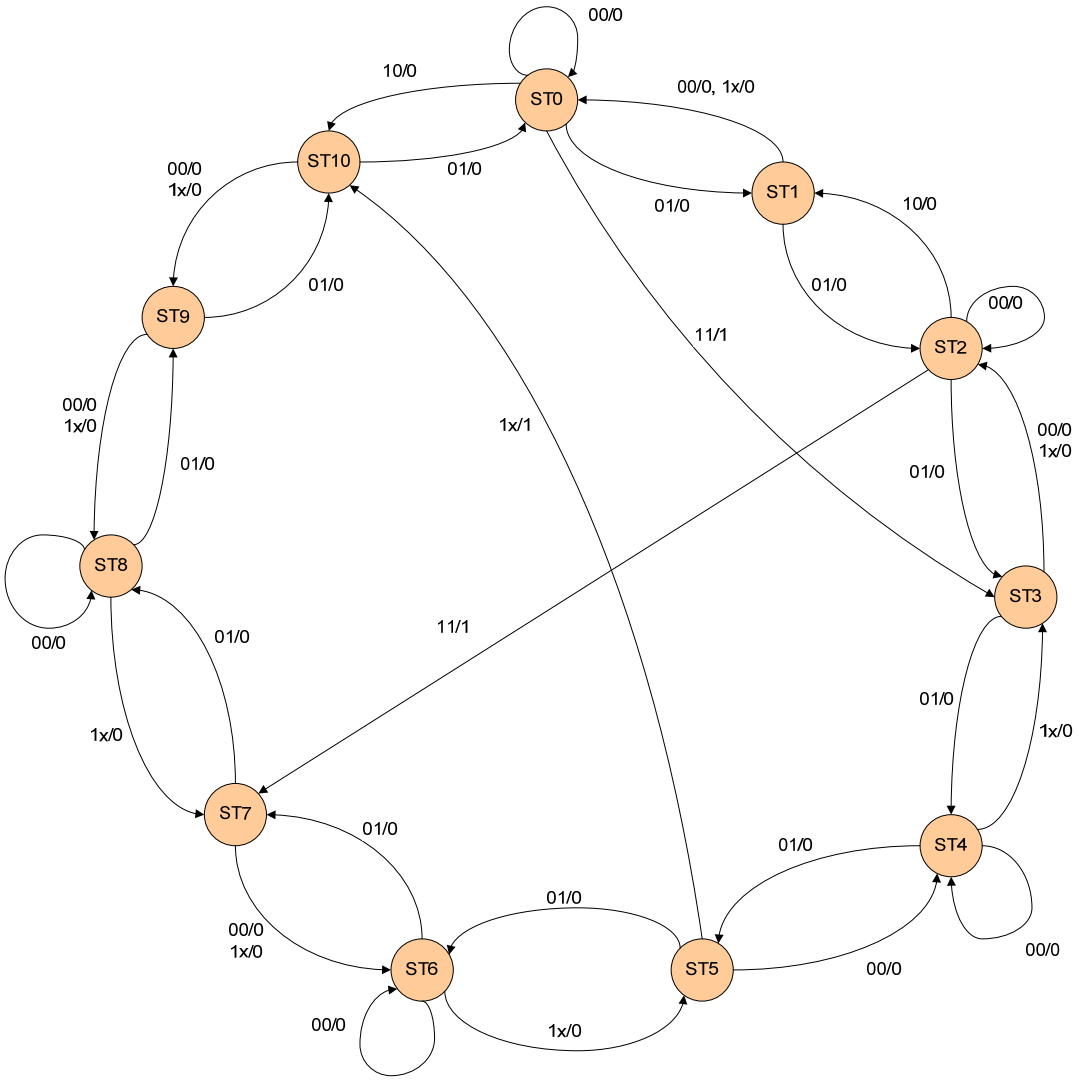
U stavových automatů se používají nejčastěji tyto možnosti kódování [50]:

- *sekvenční (binární) kódování* – každému stavu je sekvenčně přiřazena binární hodnota od 0 do n_s; používá se minimální počet k bitů,
- *kódování „one-hot“ (kódování* 1 z n_s*)* – vždy pouze jeden bit vnitřního stavu je roven log. 1, ostatní jsou v log. 0; toto kódování vyžaduje tedy n_s bitů (klopných obvodů),
- *Grayův kód* – kódování je určeno tak, aby se sousední stavy lišily pouze změnou jednoho bitu (kódová vzdálenost $d = 1$), ke kódování se používá pouze minimální počet k bitů,
- *Johnsonův kód* – obdobně jako u Grayova kódu se sousední číslice kódu liší o jediný bit, ale k zakódování n_s vnitřních stavů potřebujeme min. $n_s/2$ vnitřních bitů,
- *kódování na minimální počet bitů* – obdoba binárního kódování, ale stavy jsou voleny náhodně,
- *uživatelsky definované* – v ojedinělých případech lze zvolit kódování jednotlivých stavů ručně (pevné přiřazení), pak mluvíme o tzv. „Hard-encoded“ automatu.

O použitém kódování stavového registru rozhoduje v běžných případech sám návrhový systém (syntezátor) a volí z jeho pohledu nejvhodnější kódování. Návrhář však může syntezátoru vnutit vlastní kódování – buď speciálními příkazy HDL jazyka, nebo změnou parametrů syntezátoru. Například návrhový systém Altera Quartus II při „defaultním“ kódování zvolí sekvenční kódování, je-li počet stavů menší než pět a kódování one-hot, je-li počet stavů mezi 5 a 50. Při větším počtu stavů se zvolí Grayův kód. Také kódování one-hot v tomto návrhovém systému není čistý kód 1 z n_s, ale počáteční (reset) stav obsahuje samé

nuly a ostatní stavy mají vždy 2 bity v logické jedničce (jeden z nich je vždy MSB). Toto je vhodnější pro definované chování automatu při zapnutí napájení. Obecně je z hlediska úspory klopných obvodů lepší používat kódování s menším počtem bitů (sekvenční nebo grayův kód). U FPGA obvodů, které mají dostatek registrů, je spíše tendence používat kódování one-hot, které vede na vhodnější mapování přechodové a výstupní funkce do LUTů (signály se příliš nevětví). U one-hot kódování se rovněž mezi sousedními stavy překlápí pouze 2 bity, což vytváří méně hazardů a může mít u větších automatů i vliv na spotřebu energie.

Pokud pro počet vnitřních stavů automatu platí rovnost ve vztahu (4.1), mluvíme o **spolehlivém** (safe) stavovém automatu (nemá žádný nevyužitý stav). Tento automat většinou využívá binární nebo Grayův kód. V případě nerovnosti ve vztahu (4.1) mluvíme o **nespolehlivém** (unsafe) automatu. Pokud ošetříme všechny nevyužité vnitřní stavy (za cenu složitější přechodové funkce), získáme spolehlivý automat. Požadavek na syntézu spolehlivého automatu lze v návrhových systémech nastavit vhodnými parametry (např. v systému Quartus aktivovat volbu na „Safe State Machine“). Pokud nepreferujeme



Obr. 30: Testovací stavový automat

spolehlivý automat, je syntéza optimalizována na maximální výkonnost (rychlost) a méně logiky, a neošetřují se nevyužité stavy.

Podobně jako optimální volba kódování, není jednoznačná ani odpověď na otázku, zda je výhodnější používat stavové automaty typu Mealy nebo Moore. Mooreho automat většinou vede na vyšší počet vnitřních stavů, ale naproti tomu má lépe syntetizovatelnou logiku s kratšími kritickými cestami. Obecně se doporučuje v FPGA obvodech volit u složitějších automatů s více vstupy a výstupy raději Mooreho automat, z podobných důvodů jako jsme zdůvodňovali volbu one-hot kódování [51]. Nevýhodou Mealyho automatu je změna výstupu ihned se změnou vstupu, což může způsobovat statické hazardy. Proto se někdy opatřují výstupní signály dalšími registry, které ale zvyšují latenci signálů. Vše samozřejmě ještě souvisí s konkrétní hardwarovou architekturou použitého FPGA obvodu.

Pro ověření předchozích úvah byl navržen a v návrhovém systému Altera Quartus II syntetizován konkrétní stavový automat (do FPGA řady Cyclone III), jehož diagram je na obr. 30. Výsledky jsou uvedeny v tab. 2. Automat byl popsán v jazyce VHDL ve dvou variantách – typ Mealy a typ Moore (vstupní soubory obou automatů jsou v příloze práce). Mealyho automat měl 11 vnitřních stavů, Mooreho automat s toutéž funkcí vedl na 14 stavů. Automat měl 2 vstupní bity (4 vstupní stavy) a jeden výstupní bit. Výsledky v tabulce vypovídají o množství zabrané logiky. Je z nich patrný výrazný nárůst logiky pro spolehlivé automaty především u one-hot kódování. Z pohledu zabrané logiky se nejeví one-hot kódování příliš výhodné. Z tohoto relativně jednoduchého příkladu nelze však činit obecnější závěry.

Tab. 2: Srovnání implementace různých variant stavových automatů

Typ automatu		Kódování	Počet log. elementů	Počet registrů	Max. hodinový kmitočet [MHz]
Mealy	Nespolehlivý	One-hot	21	11	250
		Sekvenční	20	4	250
		Grayovo	22	4	250
	Spolehlivý	One-hot	33	11	250
		Sekvenční	24	4	250
		Grayovo	27	4	250
Moore	Nespolehlivý	One-hot	21	14	250
		Sekvenční	24	4	250
		Grayovo	25	4	250
	Spolehlivý	One-hot	41	14	177
		Sekvenční	24	4	250
		Grayovo	24	4	250

4.3.3 Výkonnost implementovaných algoritmů

Jak již bylo naznačeno, kvalita vlastního digitálního návrhu se vesměs posuzuje podle tří kritérií – maximální rychlosti, zabrané plochy (logiky) a elektrické výkonové spotřeby. V FPGA obvodech se někdy výkonová spotřeba nebere v úvahu a **efektivita** navržených algoritmů se posuzuje podle hodnoty součinu času na zpracování dané úlohy a množství zaplněné logiky (množství LUT, registrů, příp. bloků RAM a DSP bloků). Převrácená hodnota času zpracování dané úlohy T_u se označuje jako **výkonnost** P:

$$P = 1 / T_u \tag{4.2}$$

V procesorových systémech jsme nejčastěji porovnávali výkonnost v jednotkách MIPS (Million Instructions Per Second, milióny instrukcí za sekundu), což je u hardwarových systémů nepoužitelné. Čas na zpracování jedné úlohy T_{CPU} byl v procesoru dán výkonnostní rovnicí [52]:

$$T_{CPU} = IC \,.\, CPI \,.\, T_{clk} \tag{4.3}$$

kde IC je počet instrukcí dané úlohy (Instruction Count), CPI je průměrný počet taktů na instrukci (Cycles Per Instruction) a T_{clk} je použitá perioda hodinového kmitočtu. Pro hardwarové systémy můžeme rovnici (4.3) upravit na tvar:

$$T_u = N_T \,.\, T_{clk} \tag{4.4}$$

kde N_T je počet potřebných hodinových taktů na zpracování jedné úlohy, resp. jednoho vzorku. Ze vztahů (4.2) a (4.4) pak vyplývá vztah (4.5) pro výkonnost implementovaných algoritmů - *počet vzorků signálu zpracovaného za určitý čas*. Jednotkou je nejčastěji MSPS (Million Samples Per Second) a tuto hodnotu vypočteme podílem maximální hodinové frekvence a počtem hodinových taktů na jeden vzorek:

$$P = f_{clk} / N_T \tag{4.5}$$

Tato veličina se nepoužívá pouze u FPGA obvodů, ale u všech systémů zpracovávajících číslicový signál (např. u DSP procesorů). Dalším významným parametrem pro porovnávání výkonnosti v oblasti zpracování digitálních signálů je *maximální teoretický výkon* v jednotkách MMAC/s (Million Multiply Accumulates Per Second, milióny násobení a mezisoučtů za sekundu). Toto číslo se u FPGA obvodů získává jako počet násobiček (většinou 18 x 18 bitů) na čipu násobený maximální hodinovou frekvencí násobičky a pohybuje se v rozmezí řádově od 10^1 do 10^6. Například momentálně nejvýkonnější FPGA firmy Altera (Stratix EP4SE680) obsahuje 1360 násobiček 18 x 18 bitů a může pracovat s frekvencí 550 MHz, jeho výkonnost je tedy 748 GMAC/s (Xilinx dokonce nabízí FPGA Virtex-6 s výkonností přesahující 1 TMAC/s). Je zřejmé, že tyto výkonnosti jsou maximální možné a praktický návrh je většinou plně nevyužije.

Někdy se pro účely porovnávání kvality implementovaných algoritmů v FPGA používá parametr, který vypočteme jako podíl počtu zabraných logických buněk v FPGA obvodu a maximální vzorkovací frekvence f_{max} pro daný obvod.

4.4 Způsoby snižování napájecího příkonu

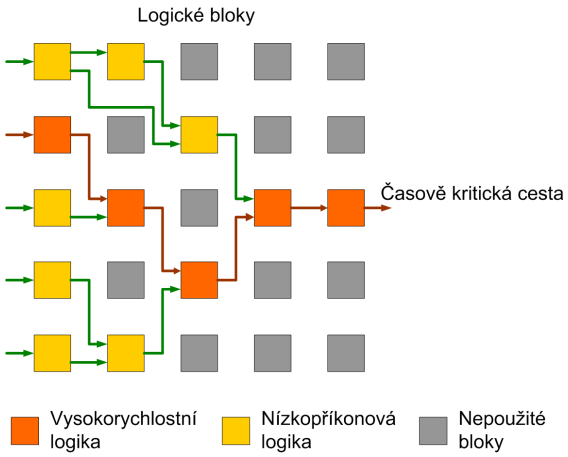
Ještě jsme se nezmiňovali o třetí charakteristice, která určuje kvalitu číslicového návrhu, tj. o výkonové energetické spotřebě. Pro snížení výkonové spotřeby máme v principu dvě cesty – technologickou a architektonickou. Technologická cesta závisí především na výrobní technologii, která určuje jednak klidovou spotřebu a jednak parametry ovlivňující dynamickou spotřebu – parazitní kapacity hradel tranzistorů, spojovacích cest, spínacích tranzistorů, velikost napájecího napětí apod. Pro další úvahy ještě předpokládejme jednu vlastnost související s technologií – nevyužitá logika FPGA obvodu je odpojená a neovlivňuje statickou ani dynamickou spotřebu. Všímejme si dále architektonických řešení, které může běžný systémový návrhář ovlivnit. Cesty ke snížení spotřeby v podstatě závisí na úrovni abstrakce systémového návrhu, na které se návrhář pohybuje.

Na úrovni návrhu architektury může návrhář navrhnout taková řešení, která spotřebují minimum logických buněk a registrů – budou se preferovat sériové sekvenční algoritmy a maximální sdílení systémových prostředků. Tím se sice uspoří plocha a s ní související statická i dynamická spotřeba, ale výrazně se sníží výkonnost celého systému. Naopak techniky proudového zpracování (pipeline) či paralelního zpracování dat vedou na rozsáhlejší logiku, tím pádem i na větší proudovou spotřebu. Pokud bychom ale předpokládali u obou zmiňovaných technik dosažení stejné výkonnosti, stačí paralelní architekturu taktovat nižší pracovní frekvencí. Systém s větším počtem logických prvků, ale taktovaný výrazně nižší frekvencí, může v důsledku vést ke snížení celkové výkonové spotřeby.

Při rozboru snížení spotřeby na nižší úrovni abstrakce můžeme vycházet ze vztahu (2.6). Velikost napájecího napětí je dána zvoleným FPGA obvodem (uvažujeme především napětí vlastního logického jádra), jednotkové parazitní kapacity určuje použitá výrobní technologie. Návrhář, resp. návrhový systém, může částečně ovlivnit délky spojových cest, množství propojovacích matic v cestě signálu, množství větvení signálu a s tím související počet spojených hradel tranzistorů aj. Asi nejlépe ovlivnitelnou veličinou ve vztahu (2.6) je pracovní frekvence. Většina současných návrhů systémů je plně synchronních a registrově orientovaných. Případná blokace klopných obvodů se řeší signály „enable“, které jsou v architekturách hradlových polí běžně implementovány. Tím nevkládáme do cesty hodinového signálu žádnou logiku (zpoždění), a přitom nedochází ke změnám logických úrovní klopných obvodů. Pokud potřebujeme „zastavit“ hodinové signály do větší části obvodu (časové domény), řešíme to přes již zmiňované hodinové manažery (clock management), které jsou dnes běžnou součástí moderních FPGA obvodů. Používání

asynchronních klopných obvodů nebo hradlování hodinových signálů by sice mohlo přinést návrháři částečné snížení spotřeby, ale za cenu nepřijatelného snížení spolehlivosti systému.

Jedním ze způsobů snížení spotřeby, který lze zařadit na pomezí technologických a architektonických řešení, je tzv. *programovatelná technologie řízení spotřeby* (Programmable Power Technology), zavedená před několika lety firmou Altera v obvodech FPGA řady Stratix [53]. Tato technologie umožňuje konfigurovat logické buňky do dvou režimů (obr. 31) - buď jsou buňky rychlé a s vyšší spotřebou (tzv. standardní mód) nebo jsou nízkopříkonové se zpomalením funkce logiky. Změna módu se dosahuje změnou napětí substrátu u CMOS tranzistorů, čímž se ovlivňuje velikost jejich prahového napětí a následně svodový proud. Volbu rychlosti a spotřeby buněk v podstatě provádí automaticky sám návrhový systém a nevyžaduje zásah návrháře. Rychlá logika se použije jen v místech kritických cest, kterých je v typickém návrhu přibližně 20 % [53]. Nízkopříkonový režim logiky šetří 50 % energie a používá se nejen u logických buněk, ale také u DSP bloků a u paměťových bloků.



Obr. 31: Programovatelná technologie řízení spotřeby

Jinou zajímavou technologicko-architektonickou metodou snížení spotřeby je použití dvouhranových klopných obvodů – máme na mysli dvouhranové obvody vytvořené v architektuře hradlových polí jako primitivum (neuvažujeme složení dvouhranového klopného obvodu ze dvou jednohranových, jak je v systémech časté). Příkladem může být tzv. *technologie CoolClock* použitá v obvodech Xilinx Coolrunner-II. Při syntéze se vydělí vstupní frekvence dvěma a použijí se zmiňované dvouhranové obvody. Tím se v podstatě pracovní frekvence sníží na polovinu bez ztráty výkonnosti.

Nezanedbatelný podíl na celkové výkonové ztrátě mají I/O buňky. Jejich spotřebu nejvýrazněji ovlivňují proudy tekoucí výstupními piny. Tyto proudy jsou jednak dány přechodovými ději při změně logické úrovně a jednak statickými proudy způsobenými odporovou zátěží (signálové piny zakončené terminátory, pull-up rezistory v otevřených kolektorech apod.). Návrhář může ovlivnit zmiňované ztráty vhodnou volbou I/O standardů, optimální velikostí pull-up rezistorů, preferováním sériových terminátorů vedení před

paralelními, či vhodnou strmostí náběžných/sestupných hran výstupních signálů. Z hlediska vstupních signálů má u vesměs používané technologie CMOS největší vliv na spotřebu I/O buněk rychlost náběžných/sestupných hran, tj. doba přechodového děje. Důležité je také připojení všech nepoužitých vstupních pinů na definovanou logickou úroveň (nenechat je plovoucí).

4.5 Pravidla synchronního návrhu

Se vzrůstající velikostí číslicových obvodů roste i nebezpečí vzniku obvodů s nedeterministickým chováním. Přestože moderní návrhové systémy na mnohé problémy upozorní, je automatizován pouze návrh synchronních obvodů. Synchronním obvodem rozumějme obvod, ve kterém se mění stavy všech registrů současně, vesměs v důsledku změny hodinového signálu. V jednom číslicovém obvodu však může být více hodinových signálů. Potom ta část obvodu, která je taktována jediným hodinovým signálem, se nazývá **časová doména** (clock domain). Signál, který se mění pouze v důsledku změny hodinového signálu, nazývejme *synchronní*. Signál, který se může měnit kdykoli, označujme jako *asynchronní*.

Právě asynchronní signály jsou významným problémem číslicových systémů. Typickým příkladem asynchronního signálu jsou externí vstupy do FPGA obvodu, které se obecně mohou měnit kdykoli (asynchronně vůči interním hodinovým signálům). Problémy mohou způsobit i asynchronní resety či sety – při nedefinovaném ukončení jejich aktivní úrovně. Se zpracováním asynchronních signálů se také běžně setkáváme při implementaci nejrůznějších komunikačních rozhraní (např. RS232, SPI, I2C).

Rovněž na přenos signálů mezi dvěma časovými doménami (tzv. *clock domain crossing*) jsou obecně asynchronní – hodinové signály v jedné doméně jsou asynchronní vůči hodinám v druhé doméně. Samozřejmě může nastat případ, že oba hodinové kmitočty v doménách jsou vzájemně synchronní, potom lze implementaci mezidoménových přechodů signálů výrazně zjednodušit. V dalším textu budeme předpokládat obecný případ přenos signálů mezi asynchronními doménami.

U složitých číslicových systémů se lze běžně setkat s více časovými doménami a je nutné si vyjasnit, které signály jsou synchronní a které asynchronní. Chyby v návrhu s asynchronními signály jsou zejména u začátečníků velmi časté a můžou způsobit nepředvídatelná selhání obvodu, která se velmi obtížně hledají. Celý systém se pak chová nespolehlivě a nahodile. Popisované jevy se navíc nemusí odhalit při ověřování a testování systému a mohou se projevit až za provozu. Podívejme se, co je nejčastější příčinou nespolehlivosti a následně, jakým způsobem je možné ji odstranit.

4.5.1 Metastability

Nespolehlivé chování je ve většině případů způsobeno tzv. *metastabilitami registrů*. Metastabilitou nazýváme neschopnost výstupu registru ustálit se na definované logické úrovni v přesně definovaném čase, obvykle za jednu hodinovou periodu [54]. U výstupu registru v metastabilním stavu nelze předpovědět jeho chování. Podle použité technologie se metastabilita může projevovat jako delší výskyt nedefinované logické úrovně mezi log. 0 a 1, jako krátký zákmit, oscilacemi nebo větším zpožděním registru. Nedefinovaná logická úroveň může v technologii CMOS způsobit otevření obou tranzistorů (s kanálem P i N) a způsobovat tak proudové špičky v napájecím obvodu. Je-li za metastabilním výstupem registru zařazeno více dalších logických členů, může každý z nich vyhodnotit na svém vstupu jinou logickou hodnotu. Asynchronní signál v logice stavového automatu může způsobovat metastabilní chování na výstupu stavového registru. Automat pak může přejít do náhodného nebo nedefinovaného stavu a „zamrznout“. Na metastabilní chování má vliv i rychlost změny vstupního datového signálu, strmost hodinového signálu, velikost a kvalita napájecího napětí, rychlost logických prvků v obvodu či teplota čipu.

Příčinou metastabilního chování registru je porušení jeho časových parametrů – *doby předstihu* t_s (setup time), *doby přesahu* t_h (hold time), případně *doby zotavení po resetu* t_{rr} (reset recovery time). Nejčastěji při běžné činnosti systému dochází k metastabilitám v okamžiku, kdy se při příchodu aktivní hrany hodinového signálu mění současně signál na datovém vstupu registru (v průběhu dob t_s a t_h). Proto by měl návrhář systému věnovat dodržování těchto časů náležitou pozornost. Určité problémy může u plně synchronních systémů naznačit důkladně provedená číslicová simulace s vhodně volenými stimuly a statická časová analýza. Bohužel problémy s asynchronními vstupy a následné metastability simulace většinou neodhalí. Proto je třeba k návrhu přistupovat systematicky a dodržovat určitá pravidla, jak bude naznačeno níže.

Metastability se posuzují *střední dobou bezporuchového provozu* označovanou *MTBF* (Mean Time Between Failures). Tato doba udává průměrný čas mezi dvěma metastabilitami (mezi dvěma poruchami v synchronizaci) v návrhu a lze vyčíslit následujícím vztahem [55]:

$$MTBF = \frac{e^{t_{met}/C_2}}{C_1 \cdot f_{clk} \cdot f_{data}} \tag{4.6}$$

kde f_{clk} ……. hodinový kmitočet cílové časové domény,
f_{data} ……kmitočet změny dat vstupního asynchronního signálu (zdrojové domény),
t_{met} …… doba přechodu z metastabilního stavu do definované logické hodnoty,
C_1 a C_2 …. konstanty závislé na typu obvodu a na použité výrobní technologii.

Bohužel zjistit hodnoty C_1 a C_2 pro konkrétní řady obvodů FPGA je problematické a výrobci je běžně neuvádějí. Pouze v [56], [57] a [58] byly nalezeny hodnoty C_1 a C_2 pro starší

typy FPGA obvodů – viz tab. 3. Z tabulky je patrné, že se uvedené konstanty (zejména C_1) pohybují relativně v širokých mezích. Proto, pokud nejsou známé pro konkrétní FPGA, je obtížné počítat *MTBF*.

Tab. 3: Hodnoty konstant C_1 a C_2 vybraných FPGA obvodů

Typ FPGA	C_1 [s]	C_2 [s]
Actel ACT1	$1{,}00.10^{-9}$	$2{,}17.10^{-10}$
Actel ProASIC3/E	$9{,}11.10^{-12}$	$6{,}37.10^{-11}$
Altera MAX7000	$2{,}98.10^{-17}$	$2{,}00.10^{-10}$
Altera FLEX10K	$1{,}01.10^{-13}$	$7{,}89.10^{-11}$
Xilinx XC3020-70	$1{,}50.10^{-10}$	$2{,}71.10^{-10}$
Xilinx XC8100	$2{,}15.10^{-12}$	$4{,}65.10^{-10}$
QuickLogic QL12x16-0	$2{,}94.10^{-11}$	$2{,}91.10^{-10}$
QuickLogic QL12x16-2	$1{,}23.10^{-10}$	$1{,}85.10^{-10}$

Ze vztahu (4.6) vyplývá, že *MTBF* je nepřímo úměrná rychlosti změn na vstupu a pracovnímu kmitočtu dané časové domény. Obecný trend stále se zvyšujících pracovních frekvencí tedy vede k neustálému snižování času mezi dvěma poruchami. Na snižování *MTBF* má však mnohem výraznější vliv snižování napájecího napětí logických obvodů, které se projevuje výraznou změnou konstant C_1 a C_2 (viz níže). Vztah (4.6) navíc udává *MTBF* pouze pro jeden „styk“ dvou časových domén. V návrhu je ale možných přenosů signálů mezi časovými doménami většinou více, obecně N_C. Poruchovost celého návrhu Q_C je pak dána vztahem [55]:

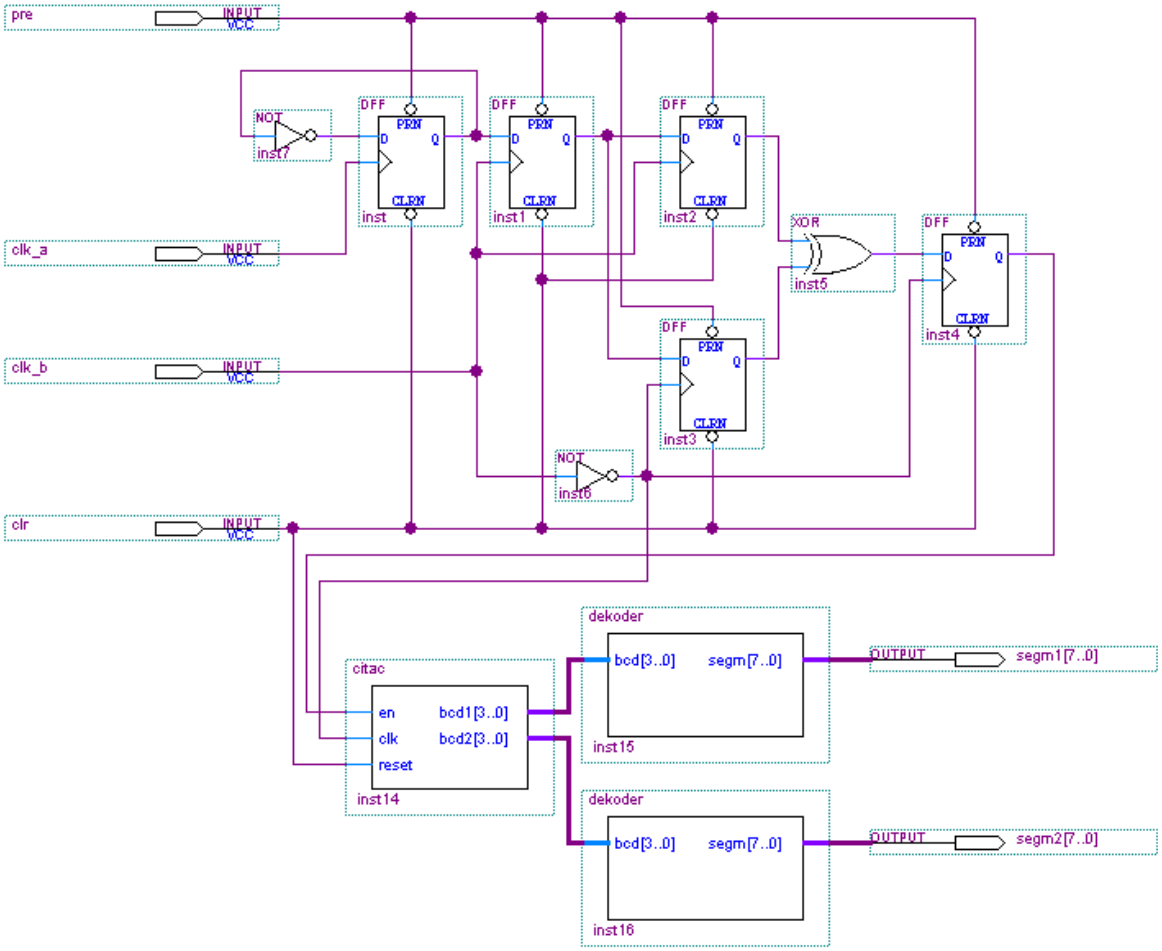
$$Q_C = \frac{1}{MTBF_C} = \sum_{i=1}^{N_C} \frac{1}{MTBF_i} \tag{4.7}$$

kde $MTBF_C$ je střední doba bezporuchového provozu celého návrhu. Ze vztahu (4.7) je zřejmé, že $MTBF_C$ nemůže být nikdy vyšší než je *MTBF* nejhoršího styku dvou časových domén. Optimální tedy je, pokud jsou $MTBF_i$ v celém návrhu srovnatelné a je jich co nejméně. Moderní návrhové systémy umí celkové $MTBF_C$ spočítat (odhadnout) na základě identifikace synchronizačních řetězců, časových domén a statické časové analýzy navrženého systému.

Uvedený vztah (4.6) jsme se snažili ověřit na logice FPGA obvodů. Vlivem průchodu signálu I/O buňkami a neodhadnutelnému zpoždění uvnitř obvodu se podle očekávání ukázalo testování dosti problematické. U samotného klopného obvodu typu D, na jehož datovém

vstupu se mění signál současně s aktivní hranou hodinového signálu, nebylo naměřeno žádné metastabilní chování. Pravděpodobně by bylo nutné zajistit drobný předstih nebo přesah obou signálů.

Dále jsme zkoušeli změřit metastabilní chování u zapojení se dvěma nesoudělnými frekvencemi (časovými doménami). Použili jsme testovací obvod podle [55], jehož upravené schéma je na obr. 32. Popišme si stručně jeho funkci. Na vstupu je z klopného obvodu *inst* a



Obr. 32: Testovací obvod pro zjišťování metastabilit

invertoru vytvořena dělička dvěma, která zajišťuje na svém výstupu změnu logické úrovně s každou aktivní hranou hodinového signálu *clk_a*. Následuje synchronizační buňka (její účel je popsán v kap. 4.5.2) složená z klopných obvodů *inst1* a *inst2* a taktovaná druhým hodinovým signálem *clk_b*. Signál za prvním klopným obvodem *inst1* této buňky je strobován v polovině periody *clk_b* dalším klopným obvodem *inst3* a následně porovnáván hradlem XOR s výstupem synchronizační buňky. Případný rozdíl je zachycen v registru *inst4*. Následující bloky slouží jen k čítání a zobrazení chyb na sedmisegmentovém displeji.

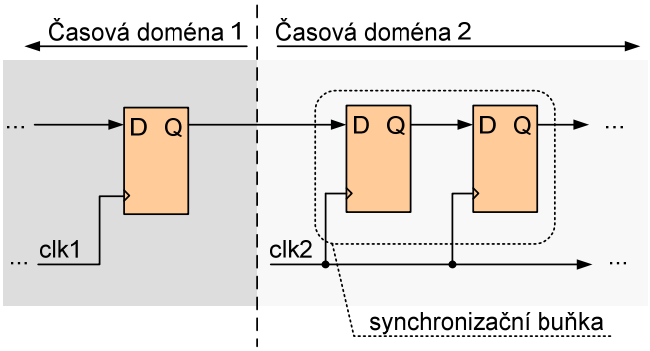
Snahou bylo pokusit se určit konkrétní hodnoty konstant C_1 a C_2 ve vztahu (4.6). Nejprve jsme testovali starší hradlové pole v 5voltové logice jádra – konkrétně Altera EPF10K20RC240-3. Dosazením do (4.6) při použití konstant z tab. 3 pro maximální

dosažitelné kmitočty f_{data} a f_{clk} do 25 MHz vychází *MTBF* v řádech stovek let. Praktické měření toto potvrdilo - na tomto obvodu jsme během několika dní testování nenaměřili žádné metastabilní chování.

Jiná byla situace u FPGA s 1,2-voltovou logikou jádra – typ Altera EP3C5E144C8N. Zde byly metastability měřitelné, jejich počet byl ale silně nereprodukovatelný (nárazový až náhodný) a se změnou jedné z frekvencí se přiměřeně neměnila *MTBF*. Proto bylo od identifikace konstant C_1 a C_2 upuštěno. Významným závěrem je ale skutečnost, že *MTBF* se výrazně snižuje při snižujícím se napájecím napětím, čímž je problém metastabilit pro návrháře stále aktuálnější.

4.5.2 Odstraňování metastabilit

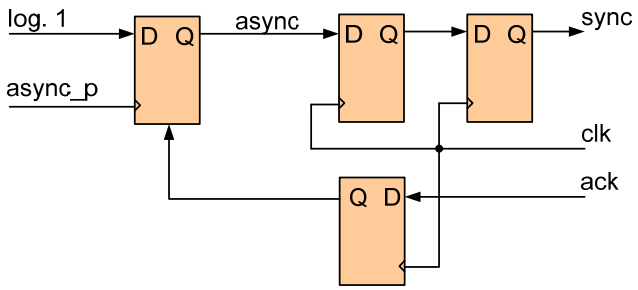
Jak jsme již naznačili, metastabilní stav na výstupu registru po určité době odezní a klopný obvod se překlopí do definované logické hodnoty 0 nebo 1 (konkrétní hodnota není důležitá). Pokud se tedy po určité době bude výstup registru opětovně strobovat, již by měl mít definovanou logickou úroveň. Eliminace metastabilního stavu tedy vychází z myšlenky, že doba následného snímání je rovna periodě hodinového signálu příslušné časové domény. Proto se pro odstranění (lépe řečeno pro výrazné omezení) metastabilit používá obecně řetězec synchronizačních registrů nazývaný **synchronizátor** (synchronizer, synchronizační buňka). Nejčastěji je synchronizátor tvořen dvojicí registrů (obr. 33), výjimečně lze pro zvýšení *MTBF* použít i trojici. Pokud za prvním registrem synchronizátoru dochází k metastabilnímu chování, předpokládáme, že během periody hodinového signálu dojde k ustálení jeho výstupu a druhý registr pak navzorkuje již definovanou logickou úroveň.



Obr. 33: Dvojitý synchronizátor pro eliminaci metastabilit

Určitou nevýhodou synchronizátoru je jeho „nejednotné“ zpoždění, které za daných okolností může být jeden, dva, a dokonce až tři hodinové takty. Dalším problémem je, že synchronizátor může filtrovat rychlé nebo krátké změny vstupního signálu. Obecně u periodických signálů je třeba splnit podmínku $f_{data} \leq f_{clk}$. Obtížně řešitelná je situace, kdy $f_{data} > f_{clk}$. Pokud v tomto případě nechceme ztratit vstupní informace, je vhodné přistoupit k používání zpětné vazby (handshake). Tím zajistíme, že se informace na vstupu neztratí ani

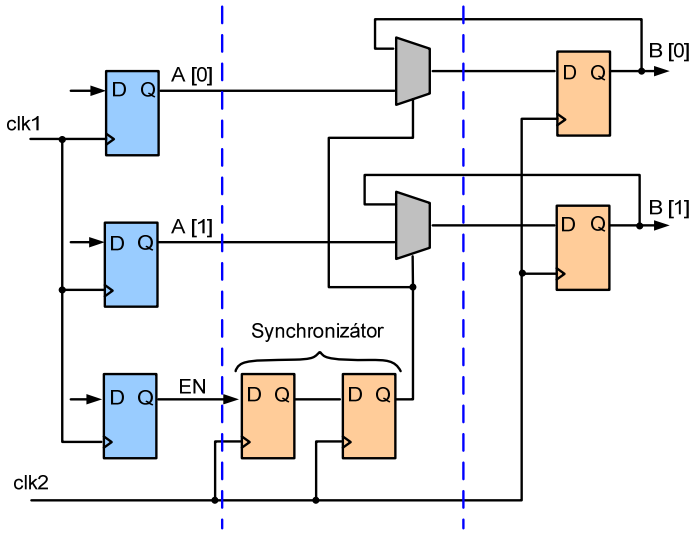
nezduplikují. Zmiňovaná zpětná vazba však vnáší do signálové cesty další doplňkové zpoždění.



Obr. 34: Obvod pro detekci a synchronizaci impulsů

Pokud potřebujeme zachytávat impulsy užší než je perioda hodinového signálu (ale pomalu se opakující), je možné použít zapojení na obr. 34 [54]. Vstupní asynchronní signál *async_p* je připojen na hodinový vstup záchytného registru, na datovém vstupu je trvalá log. 1. Tento vstupní registr je opět doplněn synchronizátorem. Po úspěšném zpracování události je možné signálem *ack* vstupní záchytný registr vynulovat. Protože signál *ack* slouží jako asynchronní reset registru, je třeba k eliminaci logických hazardů a metastabilit rovněž zajistit jeho synchronnost. Doporučuje se proto použití dalšího registru. Nevýhodou popisovaného obvodu je nemožnost detekovat události rychlejší, než jsou čtyři periody hodinového signálu – v nejhorším případě totiž potřebujeme tři periody hodin na převzorkování asynchronního signálu a jednu periodu na synchronizaci signálu *ack*.

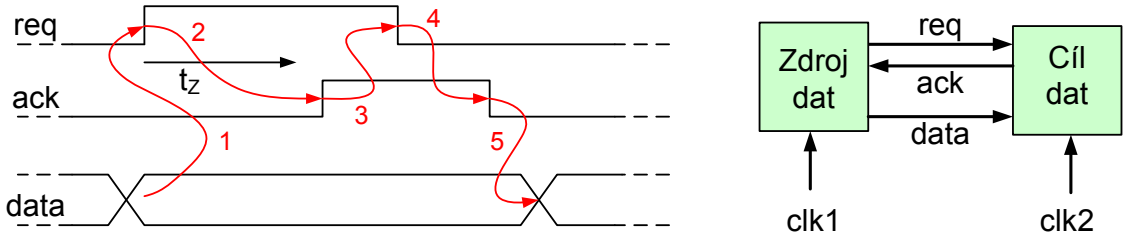
Dosud jsme uvažovali o eliminaci metastabilit vznikajících při přenosu jednobitových signálů. Složitější situace nastává, převádíme-li z jedné časové domény do druhé vícebitový (obecně *N*-bitový) signál. Obecně nelze použít *N* samostatných synchronizačních buněk, protože zpoždění na jednotlivých bitech nemusí být (zvláště v FPGA obvodech) stejná. Pokud



Obr. 35: Synchronizace vícebitové sběrnice

na sběrnici mění v jeden okamžik více bitů najednou svou logickou hodnotu, může se ve druhé časové doméně vyhodnotit na minimálně jednu periodu hodinového signálu na sběrnici nesprávná hodnota. Řešení by mohla být například dodatečná podmínka, že se na *N*-bitové sběrnici nesmí nikdy měnit více než jeden bit. Toto je však většinou nepřijatelná podmínka, a proto výhodnějším řešením je doplnění sběrnice o další jednobitový signál, který určuje platnost dat na sběrnici po jejich změně [59] – viz obr. 35. K povolení zápisu do registrů v cílové doméně je zde použit signál *EN*, který ale musí být z důvodů metastabilního chování rovněž přesynchronizován do cílové časové domény (použitím synchronizátoru). Ve zdrojové časové doméně je však nutné zajistit stabilitu sběrnice (signály *A[0]* a *A[1]*) jak po dobu aktivního signálu *EN*, tak po dobu ještě minimálně dalších tří period signálu *clk*2. Frekvence změn ve zdrojové časové doméně musí být tedy nižší, než je hodinová frekvence cílové domény. Vlastní datové vodiče sběrnice není třeba synchronizovat, neboť se při zápisu do výstupních registrů nemění.

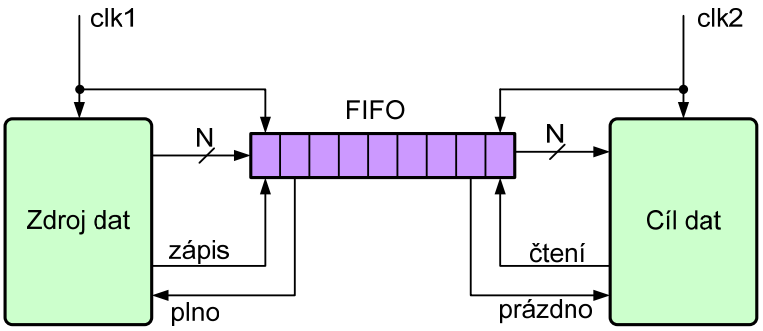
Pokud by se data na sběrnici měla měnit rychleji, než je hodinová frekvence cílové časové domény, je třeba přistoupit (obdobně jako u jednobitových signálů) ke korespondenčnímu režimu. Příklad takovéto komunikace včetně časových průběhů řídicích signálů ukazuje obr. 36 [54]. Kromě vlastních datových vodičů sběrnice zde jsou další dva signály – požadavek na přenos dat *req* (request) a potvrzení zpracování dat *ack* (acknowledge). Zdrojová strana připraví data a aktivuje signál *req*, cílová strana začne zpracovávat data a jako odpověď po zpracování aktivuje signál *ack*. Zdrojová strana následně deaktivuje signál *req*. Reakcí na to deaktivuje cílová strana signál *ack*, a tím může zdroj dat připravit na sběrnici nová data. Pro eliminaci metastabilit je třeba signály *req* i *ack* do příslušné časové domény přesynchronizovat synchronizační buňkou, čímž se přenosová rychlost na sběrnici dále sníží. Vlastní datové signály sběrnice jsou v okamžiku předávání ustálené, a proto není třeba je synchronizovat.



Obr. 36: Korespondenční režim komunikace mezi časovými doménami

Přenos informací po datové sběrnici mezi časovými doménami je možné zajistit také pomocí **paměti FIFO** (obr. 37), která umožňuje taktovat vstupní a výstupní bránu samostatnými hodinovými signály (tzv. *dual clock FIFO*). Z jedné strany se do FIFO ukládají *N*-bitová data, dokud fronta nebude signalizovat zaplnění signálem *plno*. Na výstupu se data mohou libovolně odebírat, dokud není fronta prázdná (indikováno signálem *prázdno*). Paměť

FIFO není třeba vždy navrhovat, lze použít existující implementace ve formě IP jader. V některých moderních FPGA (např. Xilinx řady Virtex) jsou již přímo zaintegrovány řadiče FIFO, které pro paměťové buňky FIFO využívají blokovou RAM.



Obr. 37: Paměť FIFO pro komunikaci mezi dvěma časovými doménami

Závěrem této části bychom chtěli doporučit kontrolu nastavení nástroje syntézy v návrhovém systému při používání synchronizačních buněk. Návrhový systém může při nevhodném nastavení parametrů vyjímat (redukovat) kaskády registrů, ze kterých nejsou v jednotlivých stupních odebírány signály (u nichž je tzv. fanout roven nule). V systému Quartus firmy Altera je třeba mít správně nastaven parametr „Synchronization Register Chain Length“ (délka řetězce synchronizační buňky), který má standardně hodnotu 2.

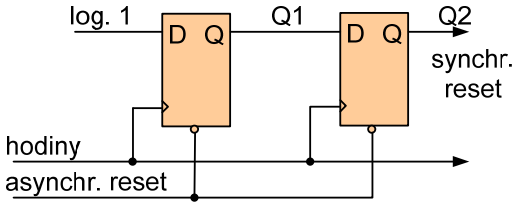
4.5.3 Další pravidla návrhu spolehlivých synchronních systémů

Jak bylo naznačeno, metastabilní chování je velmi nežádoucím jevem. Je proto třeba návrhy proti metastabilním stavům ošetřit (a to nejen použitím synchronizátorů), nespolehlivost systémů tak snížit na dostatečně nízkou hodnotu. Ze zkušenosti lze říci, že většina problémů vzniká právě ve fázi systémového návrhu. Ukažme si několik zásad a metod pro potlačení metastabilního chování, které je doporučeno dodržovat. Lze tak předejít „podivnému“ chování navržených systémů.

- *Jednotlivé bloky jsou „obaleny“ registry* – vstupní i výstupními signály bloku je doporučeno mít přímo vstupy, resp. výstupy registrů (bez další kombinační logiky). Výstupní signál tak nebude vykazovat logické hazardy a bude synchronní (bude se měnit v přesně definovaných časových okamžicích). Například i celkem běžné statické hazardy ve výstupní kombinační logice, které nejsou ošetřeny, mohou mít negativní vliv na funkci celého systému. Rovněž vstupní signály do bloku je vhodné mít zavedeny přímo do registru, což může výrazně omezit metastabilní chování - na vstupní registry je možné pohlížet i jako na první stupeň synchronizátoru. Obecně větší počet registrů v systému zkrátí kombinační cesty, a tím umožní zvýšení maximální pracovní frekvence.

- *Použití globálního resetu* – globálním resetem rozumíme nulovací signál pro všechny registry v FPGA obvodu. Jako globální reset funguje u FPGA obvodů na bázi SRAM také konfigurace obvodu, která vynuluje všechny registry. Používat tuto variantu jako globální reset je ale nevhodné a časově náročné. Implementace globálního resetu většinou nepřináší dodatečné zvětšení návrhu.

- *Synchronizace asynchronního resetu* – v případě uvolnění asynchronního resetu příliš blízko aktivní hrany hodin (pokud by současně s hranou hodin mělo dojít ke změně výstupu registru), může dojít k nedodržení doby zotavení po resetu t_{rr} a následně může nastat metastabilní chování výstupu registru. Proto je nezbytné ukončení asynchronního resetu také synchronizovat (začátek resetu není třeba synchronizovat). K tomuto účelu lze využít například synchronizátor resetu na obr. 38, který předpokládá aktivní úroveň synchronizovaného resetovacího signálu v log. 0. Je-li asynchronní reset uvolněn příliš blízko aktivní hrany hodin, může vzniknout metastabilita na signálu *Q1*, ale v podstatě ne na výstupu *Q2*. Nepatrné zpoždění ukončení resetu (o jeden hodinový takt) není na závadu. Je-li v návrhu použito více časových domén, je třeba reset přesynchronizovat do každé domény zvlášť. Řešením by mohlo být i zastavení hodinových signálů aktivací resetu (nejlépe prostřednictvím hodinových manažerů – ne použitím blokovací logiky) a jejich opětovné rozběhnutí po jeho uvolnění. Ze stejných důvodů, jak bylo v tomto odstavci naznačeno, není vhodné používat asynchronní sety a resety k běžné funkci zařízení. Dalším důvodem k omezenému používání asynchronních nastavovacích signálů je to, že v architektuře FPGA obvodů nejsou rovněž běžně k dispozici registry, které mají současně asynchronní set i reset.

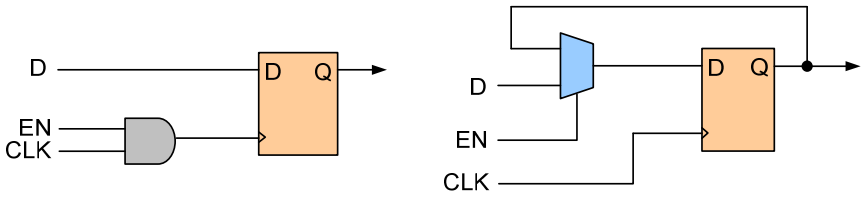


Obr. 38: Synchronizátor resetu

- *Navrhovat plně synchronní obvody* – vyvarovat se používání asynchronních sekvenčních obvodů. Asynchronní obvody v podstatě vznikají zavedením zpětné vazby v kombinačních obvodech. Takovéto struktury jsou zvláště u FPGA obvodů velmi nestabilní, nepředvídatelné a neřiditelné například v režimu testování. Častým prohřeškem zejména začátečníků je i používání signálu, který vygeneroval nějaký logický obvod, jako signál hodinový. Typickým příkladem může být kaskádování čítačů, kdy přenosový (carry) bit nižšího stupně slouží jako hodinový signál pro stupeň následující. Korektní řešení je používání vygenerovaných signálů jen jako vstupy Clock Enable (CE) synchronních klopných obvodů. Tyto klopné obvody s CE

jsou standardní součástí dnešních architektur FPGA obvodů. Další začátečnickou chybou bývá zpožďování logických signálů pomocí kaskády invertorů nebo budičů, (například za účelem „kompenzace“ zpoždění signálu v jiné větvi). Jednak je tato struktura silně závislá na konkrétní technologii a pracovní frekvenci a jednak i vlastní návrhové systémy takovéto struktury odstraňují či redukují (pokud je to nastaveno).

- *Definice hodinových signálů* – stanovit si jejich počet, aktivní hrany a vzájemné vazby mezi nimi. To umožní rozdělit obvod na jednotlivé časové domény a rozdělit jednotlivé bloky podle příslušnosti k časovým doménám, případně minimalizovat jejich počet. V každém funkčním bloku by měl být nejlépe jen jeden hodinový signál. Pokud je to jen trochu možné, je vhodné používat všude registry aktivní na stejnou hranu hodin. Hodinových signálů by mělo být co nejméně a jejich počet by neměl překročit počet globálních hodinových rozvodů v uvažovaném FPGA obvodu.
- *Nehradlovat hodinové signály* – zařazování jakékoli logiky do cesty hodinového signálu vede k problémům v synchronnosti celého systému a je třeba se tomu vyvarovat – viz obr. 39. Na tomtéž obrázku je ukázáno i doporučené řešení, které bude automaticky použito syntézním nástrojem při korektním popisu v některém



Obr. 39: Převod hradlovaných hodinových signálů na datovou zpětnou vazbu

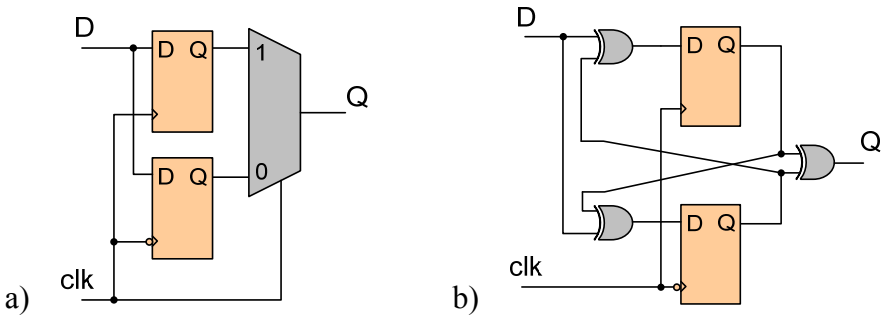
z HDL jazyků. Hradlování hodin nejen zpožďuje hodinový signál, ale nevhodným hradlovacím signálem můžou vznikat na hodinovém vstupu nepřípustně krátké impulzy. Navíc globální hodinové rozvody v FPGA nejsou určeny k připojení na vstupy kombinační logiky (na LUTy) a k propojení se musí použít standardní spoje, což vede k dalšímu zpoždění hodinových signálů, které není rovnoměrné a předvídatelné. Potřebujeme-li vypínat některým blokům v návrhu hodiny (zejména z důvodu snížení spotřeby), je vhodné k tomu použít časové manažery. V současných návrhových systémech je většinou možné využít funkce, která hradlování hodinových signálů rozpozná a konvertuje jej na použití vstupů CE v klopných obvodech.

- *Nepoužívat hladinově řízené klopné obvody* (*latche*) – preferovat používání hranově řízených klopných obvodů, které jsou sice obvodově složitější, ale změna jejich výstupu nastává v přesně daný okamžik pouze jednou za periodu hodinového signálu. Tím se logickým systémem šíří menší počet změn, což snižuje počet logických hazardů a přispívá k minimalizaci spotřebované energie. V této práci pod pojmem

registr máme na mysli vesměs hranově řízené obvody, i když to není zdůrazňováno. Tyto obvody jsou běžnou součástí architektur FPGA a podporovány jak v jazycích HDL, tak v návrhových systémech.

4.6 Implementace dvouhranových klopných obvodů a čítačů

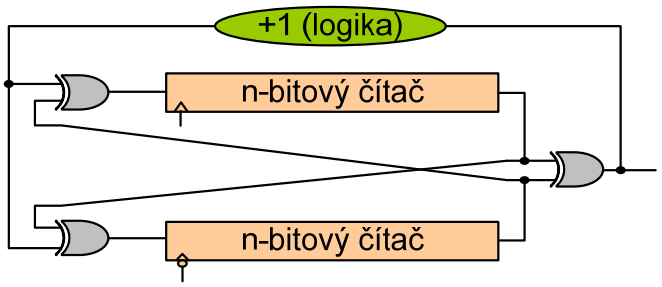
V některých případech je nutné nebo výhodné pro synchronizaci dat používat obě hrany (vzestupnou i sestupnou) hodinového signálu. Nejznámějšími aplikacemi jsou například řízení



Obr. 40: Používaná zapojení dvouhranového klopného obvodu

přenosu dat u DDR pamětí nebo kódování Manchester kódem. Problém je jednak v tom, že hardware FPGA obvodů nebývá vybaven speciálními dvouhranovými registry, a jednak ani jazyky HDL běžně nepodporují tyto obvody (jejich popis není syntetizovatelný).

Z hlediska hardwaru je proto nutné dvouhranový registr vytvořit zdvojením ze dvou běžných registrů. Každý z obou registrů je pak taktován hodinovým signálem fázově posunutým o 180°. Výstup je vybaven speciálním multiplexorem pro střídavý výběr obou signálů – viz obr. 40 a) [60]. Toto řešení je v standardních případech funkční, ale má i nevýhody. Jednak se hodinový signál dostává do kombinační logiky v datové cestě (díky vloženému multiplexeru ovládanému *clk*), což může způsobovat problémy při časových analýzách nebo při testování, a jednak je problém s použitím asynchronního resetu/setu (na výstup celého zapojení má vliv přepnutí multiplexeru). Z uvedených důvodů se častěji řeší dvouhranový klopný obvod pomocí hradel XOR, jak je naznačeno na obr. 40 b) [60]. V příloze uvádím možný popis dvouhranového klopného obvodu s asynchronním resetem/setem v jazyce VHDL.

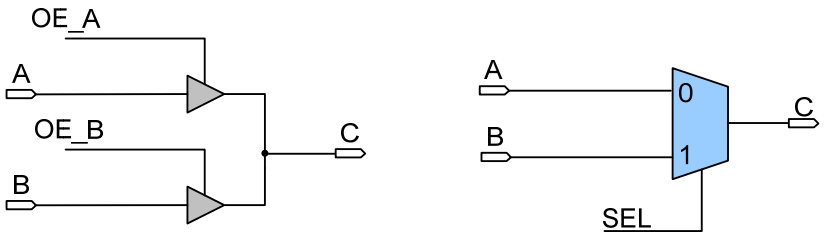


Obr. 41: Principiální zapojení dvouhranového čítače

Při své praxi jsme se setkali i s potřebou realizace dvouhranového čítače, konkrétně při přesném odměřování časových intervalů. Lze použít například obvod na obr. 41 [61], který je zobecněním zapojení na obr. 40 b). K realizaci jsou použita dvě *n*-bitová pole jednohranových klopných obvodů, načítaná hodnota je dána součtem hodnot obou polí. Je sice možné dvouhranový čítač realizovat i s výrazně menším počtem klopných obvodů (obecně *n*+1), ale opět na úkor synchronnosti a zavádění hodinového signálu do datové cesty podobně jako u popisovaného zapojení na obr. 40 a). Možný popis dvouhranového čítače v jazyce VHDL je opět uveden v příloze práce.

4.7 Implementace třístavových sběrnic

Často řešíme problém přenosu dat mezi více bloky najednou – jak uvnitř FPGA obvodu, tak s okolím. K tomuto účelu se jeví vhodné použití jednosměrných i obousměrných *třístavových sběrnic*, které mohou efektivně využít komunikační vodiče. Na druhou stranu nejsou třístavové sběrnice příliš rychlé (pomalý je zejména přechod do třetího stavu), a proto není jejich použití v některých případech výhodné – používají se převážně na nejvyšší úrovni hierarchie návrhu (top level).

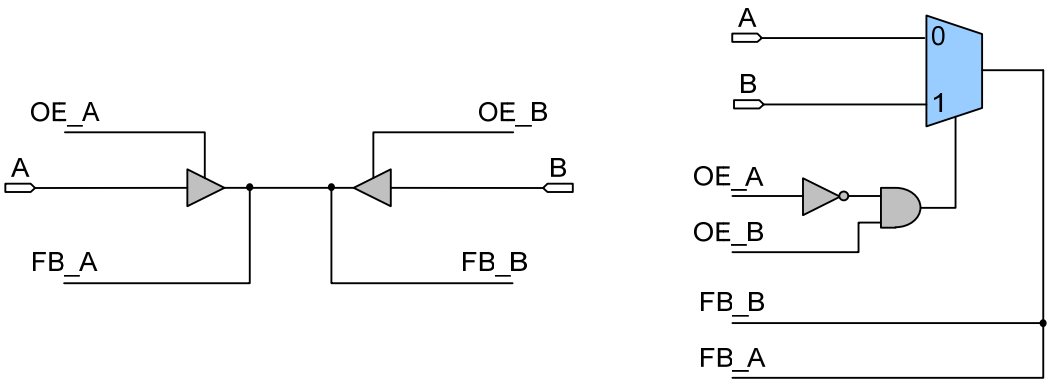


Obr. 42: Náhrada jednosměrné třístavové sběrnice multiplexorem

Hardwarová realizace třístavových sběrnic pomocí vnějších I/O buněk nebývá problematická a podporují ji v podstatě všechny CPLD a FPGA obvody. Horší může být situace při realizaci interních třístavových sběrnic. V moderních FPGA obvodech jsou již ve struktuře zaintegrovány interní jednoúčelové třístavové budiče, které syntetizátor automaticky vloží do návrhu, a tudíž se při realizaci sběrnic v podstatě nezabírá žádná další logika. V některých typech FPGA obvodů nejsou bohužel interní třístavové sběrnice podporovány. V tomto případě je třeba nahradit jednosměrnou třístavovou sběrnici multiplexorem (obr. 42), což může mít naopak výhodu v lepších dynamických vlastnostech. U více rozvětvených sběrnic však toto řešení může vést k nezanedbatelnému zabírání logiky i ke zpožďování funkce.

U obousměrných třístavových interních sběrnic (obr. 43) jsou obdobné problémy jako u sběrnic jednosměrných. Pokud uvnitř struktury nejsou třístavové budiče, je jednou z možností řešení využít třístavové budiče v I/O buňkách, a vlastní sběrnici realizovat vně FPGA obvodu [62]. Druhou možností je opět konverze na multiplexor, jak je ukázáno na obr. 43.

V jazycích HDL je (na rozdíl od hardwaru) podpora třístavových a obousměrných budičů dostatečná. Ze zkušenosti doporučuji v zápise HDL jazyka důsledně oddělovat popis logické funkce od vlastního třístavového nebo obousměrného budiče – popis budičů řešit samostatným příkazem či procesem. Je to v souladu s architekturou FPGA, kde třístavové budiče představují samostatné prvky. Příklady možného popisu třístavové jednosměrné i obousměrné sběrnice v jazyce VHDL jsou uvedeny v příloze práce.



Obr. 43: Náhrada obousměrné třístavové sběrnice multiplexorem

5 Realizace aritmetických operací v FPGA obvodech

FPGA obvody jsou stále častěji používány pro aplikace v oblasti číslicového zpracování signálu. Základem většiny algoritmů DSP je množství aritmetických operací, zejména součtů a součinů. Pro tyto operace jsou architektury FPGA stále lépe uzpůsobovány. Také nástroje pro návrh FPGA obvodů poskytují podporu pro návrh aritmetických operátorů. Přesto jejich realizace není jednoznačná a bezproblémová. Již v kap. 4.2 jsme se stručně zmiňovali o možnosti realizace aritmetických operací v procesorových jádrech, o přednostech a nedostatcích tohoto řešení. Podívejme se nyní na implementace nejpoužívanějších jednoduchých aritmetických operátorů přímo v hardwaru, v logice FPGA obvodů. Nejprve si v krátkosti shrňme problematiku kódování čísel v hardwaru, jejíž znalost je prvním předpokladem kvalitního návrhu řešení.

5.1 Kódování čísel

Pro přenos a ukládání čísel v číslicových systémech se používá dvojková číselná soustava, tj. polyadická soustava o základu 2. Libovolné číslo ve formátu **s pevnou řádovou čárkou** (FX, Fixed Point) X_{FX} lze zapsat ve tvaru:

$$X_{FX} = x_{n-1}.2^{n-1} + ... + x_0.2^0 + x_{-1}.2^{-1} + ... + x_{-m}.2^{-m} = \sum_{i=-m}^{n-1} x_i.2^i \qquad (5.1)$$

Číslo je obecně rozděleno na celou a zlomkovou část, mluvíme o tzv. *řádové mřížce* (n-1 je nejvyšší řád, m je nejnižší řád). Počet obsažených řádů v čísle nazýváme *délkou řádové mřížky* N a její velikost je dána součtem čísel n a m. Číslo zapsané v N-bitové mřížce označujeme jako *slovo* (word). Nejmenší zobrazitelné číslo nazýváme *jednotka řádové mřížky* $\varepsilon = 2^{-m}$. Nejmenší číslo, které již v řádové mřížce není zobrazitelné, označujeme jako *modul řádové mřížky* $M = 2^n$.

V rovnici (5.1) jsme uvažovali obecný tvar nezáporného binárního čísla, označovaný *unsigned* (bez znaménka). V praxi se však nejvíce používají pouze dva formáty čísla:

- *celočíselný formát* (tj. $m = 0$, $N = n$):

$$X_{FX} = x_{n-1}.2^{n-1} + ... + x_0.2^0 = \sum_{i=0}^{n-1} x_i.2^i \qquad (5.2)$$

- *zlomkový (fraction) formát* - řádová čárka umísťuje těsně za nejvyšší bit, který je řádu 2^0 ($n = 1$, $N = m+1$):

$$X_{FX} = x_0.2^0 + x_{-1}.2^{-1} + ... + x_{-m}.2^{-m} = \sum_{i=-m}^{0} x_i.2^i \qquad (5.3)$$

Z pohledu implementace aritmetických operací většinou příliš nezáleží na tom, kde je umístěna řádová čárka. Například sčítačka sečte dvě binární čísla 01001 + 00100 = 01101. Pokud na tato binární čísla pohlížíme jako celá, v desítkové soustavě součet interpretujeme jako 9 + 4 = 13. Pohlížíme-li na čísla zapsané ve zmiňovaném zlomkovém formátu, součet interpretujeme jako 0,5625 + 0,25 = 0,8125. Vždy je samozřejmě výsledek správný.

Polyadické soustavy obecně zobrazují pouze nezáporná čísla, což je pro realizaci aritmetických operací dosti omezující. Proto používáme k zobrazování záporných čísel transformace (číselné kódy), z nichž nejčastější jsou:

- *Přímý kód se znaménkem* – vznikne přidáním znaménkového bitu k absolutní hodnotě transformovaného čísla B (0 pro kladné číslo, 1 pro záporné), transformaci lze tedy zapsat:

  $$P(B) = B \quad \text{pro } B \geq 0 \quad \text{a} \quad P(B) = |B| + \tfrac{1}{2}M \quad \text{pro } B \leq 0 \tag{5.4}$$

  Vlastní transformace je jednoduchá, ale při realizaci aritmetických operací je třeba nejprve otestovat znaménko. Nevýhodou jsou dvě reprezentace nuly (např. pro 4bitové číslo: 0000 a 1000), které je nutno ošetřit.

- *Inverzní kód* – transformace tohoto kódu kladná čísla neovlivňuje, transformace záporných čísel používá tzv. jednotkového doplňku (one's complement) a vznikne negací bitů daného slova.

  $$I(B) = B \quad \text{pro } B \geq 0 \quad \text{a} \quad I(B) = M - \varepsilon + B \quad \text{pro } B \leq 0 \tag{5.5}$$

  Nevýhodou tohoto kódu je obtížnější realizace aritmetických funkcí a opět problém dvou nul (např. pro 4bitové číslo: 0000 a 1111).

- *Doplňkový kód* – transformace tohoto kódu kladná čísla opět neovlivňuje, záporná čísla se transformují aplikací dvojkového doplňku (two's complement), který vznikne přičtením ε k inverznímu kódu.

  $$D(B) = B \quad \text{pro } B \geq 0 \quad \text{a} \quad D(B) = M + B \quad \text{pro } B \leq 0 \tag{5.6}$$

  Tento kód je nejpoužívanější, neboť realizace algoritmů je nejméně problémová.

- *Aditivní kód* – jedná se v podstatě o kód s posunutou nulou – vznikne přičtením známé konstanty K k hodnotě transformovaného čísla.

  $$A(B) = B + K \tag{5.7}$$

  Pokud je posunutí rovno $K = \frac{1}{2}M$ (tj. 2^{n-1}), mluvíme o *sudém* aditivním kódu, při posunutí o $K = 2^{n-1}-\varepsilon$ mluvíme o *lichém* aditivním kódu. Výhodou tohoto kódu (na rozdíl od předchozích kódů) je, že zachovává relaci, tj. větší obraz přísluší většímu číslu. Tento kód se nejčastěji používá pro reprezentaci exponentu reálných čísel.

U všech zmíněných kódů se předpokládá, že pro transformované číslo B platí:

$$-½M \le B < ½M, \quad \text{tj. } -2^{n-1} \le B < 2^{n-1} \tag{5.8}$$

Tento omezený rozsah může vést k interpretaci nejvyššího bitu jako znaménkového, který nenese informaci o hodnotě. Takovýto pohled na nejvyšší bit ale nemusí být nejvhodnější, jak si ukážeme u nejpoužívanějšího doplňkového kódu. Definice transformace zavedená v (5.6) je rozdílná pro nezáporná a záporná čísla, což se může jevit jako komplikace. Vhodnější je definovat doplňkový kód vztahem, který platí pro celý rozsah čísel *B* podle (5.8):

$$D(B) = S_{FX} = -s_{n-1}.2^{n-1} + \sum_{i=-m}^{n-2} s_i.2^i \tag{5.9}$$

Takto definovaný tvar čísla v doplňkovém kódu chápe jako typ *signed* (se znaménkem). Tuto definici opět aplikujme na celočíselný a zlomkový formát čísla:

- *celočíselný formát*:

$$S_{FX} = -s_{n-1}.2^{n-1} + \sum_{i=0}^{n-2} s_i.2^i \tag{5.10}$$

- *zlomkový formát*:

$$S_{FX} = -s_0 + \sum_{i=-m}^{-1} s_i.2^i \tag{5.11}$$

Pro úplnost uveďme rozsahy zobrazitelných čísel v jednotlivých popisovaných formátech čísel s pevnou řádovou čárkou pro *N*-bitové slovo (tab. 4).

Tab. 4: Rozsah zobrazitelných čísel pro *N*-bitové slovo

Formát čísla / typ čísla	Bez znaménka (Unsigned)	Se znaménkem (Signed)
Obecný ($N = n + m$)	$0 \le X_{FX} \le (2^N\text{-}1).2^{-m}$	$-2^{N-m-1} \le S_{FX} \le (2^{N-m-1}\text{-}2^{-m})$
Celočíselný (N = n, $m = 0$)	$0 \le X_{FX} \le (2^N\text{-}1)$	$-2^{N-1} \le S_{FX} \le (2^{N-1}\text{-}1)$
Zlomkový ($N = m+1$, $n = 1$)	$0 \le X_{FX} \le (2\text{-}2^{-m})$ $0 \le X_{FX} \le (2\text{-}2^{-N+1})$	$-1 \le S_{FX} \le (1\text{-}2^{-m})$ $-1 \le S_{FX} \le (1\text{-}2^{-N+1})$

Dosud jsme uvažovali pevnou řádovou čárku. Popišme si i formát čísel **s pohyblivou (plovoucí) řádovou čárkou** (FP, Floating Point). Libovolné číslo v pohyblivé řádové čárce X_{FP} můžeme obecně ve dvojkové soustavě vyjádřit ve tvaru:

$$X_{FP} = M_N.2^E \tag{5.12}$$

kde M_N je mantisa čísla a E velikost jeho exponentu. Velikost bitů mantisy i exponentu si může návrhář samozřejmě volit podle daných požadavků na velikost logiky, rychlost a přesnost. Velikost řádové mřížky pro mantisu určuje přesnost vyjadřovaného čísla, rozsah

exponentu určuje rozsah zobrazitelných čísel. Rovněž typ kódování mantisy a exponentu je obecně možné volit, a každý z nich může být v jiném kódu. U čísel v pohyblivé řádové čárce se velmi často z důvodu kompatibility respektuje normalizovaný formát čísel, který zavádí mezinárodní norma IEEE 754. Nejčastěji se používají dva tvary – formát pro *jednoduchou přesnost* (single precision, 32bitový) a formát pro *dvojnásobnou přesnost* (double precision, 64bitový). Tyto dva formáty jsou natolik rozšířené, že je vhodné si je blíže popsat.

Norma IEEE 754 využívá pro zápis mantisy přímý kód se znaménkem a pro zápis exponentu lichý aditivní kód. Aditivní kód zajistí přičtením konstanty K, že transformovaná hodnota exponentu bude vždy kladná a není třeba ve formátu čísla uvádět znaménko. U formátu s jednoduchou přesností je nejvyšší bit vyhrazen pro znaménko mantisy, pak následuje 8 bitů exponentu (posunutí je tedy $K = 2^{8-1}-1 = 127$) a v nejnižších 23 bitech je uložena absolutní hodnota mantisy. U formátu s dvojnásobnou přesností je na nejvyšším bitu opět znaménko mantisy, pak následuje 11bitový exponent (posunutí je $K = 2^{11-1}-1 = 1023$) a nakonec 52bitová absolutní hodnota mantisy. Pro datový typ single může být rozsah exponentu E reálného čísla od -126 do 127, pro double pak od -1022 do 1023. Mantisa se upraví na tzv. *normalizovaný tvar*, kdy číslo začíná jedničkou (pak následuje řádová čárka a zbytek absolutní hodnoty). Do vyhrazených bitů pro mantisu nejvyšší jedničku nezapisujeme (označuje se *skrytá jednička*, hidden one).

Pokud chceme ze zápisu čísla v IEEE formátu získat zpětnou transformací reálné číslo, můžeme použít vztah:

$$X_{FP} = (-1)^{S}.2^{\exp-K}.(1, M_{IEEE}) \tag{5.13}$$

kde S je znaménkový bit mantisy, exp je hodnota exponentu v aditivním kódu ($exp = E + K$) a M_{IEEE} je absolutní hodnota mantisy bez nejvyšší jedničky. Z dosud uvedených pravidel formátu IEEE existují tři výjimky (speciální hodnoty):

- pokud $exp = 0$ a $M_{IEEE} = 0$, pak hodnota čísla je nula, resp. ±0;
- pokud exp = max. (255 nebo 2047) a $M_{IEEE} = 0$, pak hodnota čísla je ±∞ (jestliže $M_{IEEE} \neq 0$, pak je číslo neplatné, tzv. NaN);
- pokud $exp = 0$ a $M_{IEEE} \neq 0$, pak přepočtený exponent E zvětšíme o 1 a mantisu bereme ve tvaru $(0,M_{IEEE})$ – již ne ve tvaru $(1,M_{IEEE})$, protože mantisa není normalizovaná.

Abychom se vyvarovali možných chyb, je třeba mít v algoritmech tyto speciální hodnoty ošetřeny.

Součástí normy IEEE 754 je i zaokrouhlování výsledku při aritmetických operacích. Existují čtyři druhy – k nejbližšímu reprezentovatelnému číslu (implicitní), k plus nekonečnu, k mínus nekonečnu a k nule. Postup při realizaci základních algoritmů s čísly v pohyblivé řádové čárce je uveden v kap. 5.3.

5.2 Implementace základních aritmetických operátorů v pevné řádové čárce

Rozeberme si nyní realizaci základních aritmetických operací v pevné řádové čárce. Oproti FP mají na jednu stranu jednodušší implementaci, a tím zaberou méně logiky a jsou rychlejší, na druhou stranu mají menší dynamický rozsah zobrazení, a tím nižší přesnost. Ale i v realizacích s pevnou řádovou čárkou existuje mnoho různých alternativ řešení, z nichž plynou výhody, nevýhody či omezení jednotlivých implementací. U jednotlivých operací budeme naznačovat možný zápis v jazyce HDL (většinou na úrovni RTL) a případnou realizaci v FPGA.

5.2.1 Inverze znaménka

Začněme jednoduchou operací inverze znaménka, kterou často využíváme například při převodu operace sčítání na odečítání nebo k určení absolutní hodnoty čísla. Předpokládáme vstupní slovo (číslo) ve dvojkovém kódu. Ze vztahu (5.6) a z vlastností jednotkového doplňku plyne:

$$D(-a) = M + (-a) = 2^n - a - \varepsilon + \varepsilon = \mathrm{NOT}(a) + \varepsilon \tag{5.14}$$

Chceme-li tedy invertovat znaménko čísla, invertujeme všechny jeho bity a přičteme jedničku na nejnižším řádu. V jazyce VHDL bychom mohli operaci zapsat ve formě přiřazení (proměnná *a* je datového typu signed):

```
a_inv <= -a;  nebo  a_inv <= not(a) + 1;
```

Realizace v obou případech vede na sčítačku se dvěma operandy – u jednoho operandu se všechny bity obecně *N*-bitového slova invertují a druhý operand je trvale roven jedné (v nejnižším bitu je „1“). Příklad vstupního popisu v jazyce VHDL je uveden v příloze práce – testovací implementace do FPGA řady Altera Cyclone III zaplnila při *N*-bitovém datovém slovu obecně *N*-1 logických elementů (resp. jen jejich LUTů).

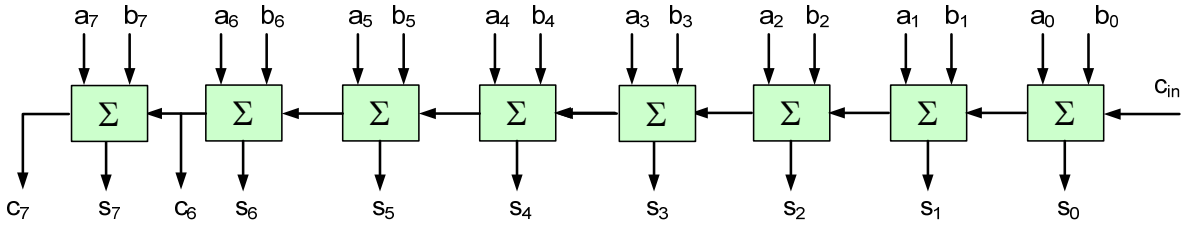
Vytváření dvojkového doplňku je možné řešit i sériovým algoritmem. Jeho algoritmus je následující – všechny bity od LSB až po první jedničku (včetně) se ponechají beze změny a zbývající vyšší bity se invertují. Možné obvodové zapojení lze nalézt např. v [46].

Je třeba si ale uvědomit, že rozsah čísel zobrazitelných ve dvojkovém kódu je asymetrický a hodnota -2^{N-m-1} nemá kladný protějšek (největší zobrazitelné kladné číslo je $2^{N-m-1}-2^{-m}$). Inverzí například 4bitového čísla 1000b vznikne opět číslo 1000b. Řešením může být buď rozšíření rozsahu o další bit, nebo alespoň detekce zmiňovaného čísla na vstupu.

6.2.2 Sčítání a odečítání

Operace sčítání je jednou z nejpoužívanějších, a je součástí i dalších složitějších algoritmů. *N*-bitová sčítačka je tak klíčovým aritmetickým obvodem a na jejích vlastnostech závisí parametry celého systému pro číslicové zpracování signálu. Ve většině případů používáme sčítačky se dvěma operandy (budeme je uvažovat v následujícím textu), ale v některých případech lze zvolit i sčítačky víceoperandové [63]. Sčítačky lze obecně rozdělit na sériové a paralelní.

Obvodově nejjednodušší je *sériová sčítačka*, která se v podstatě skládá z jednobitové úplné sčítačky doplněné jedním klopným obvodem pro uchování přenosu do vyššího řádu. Pro sečtení *N*-bitového čísla je v podstatě třeba *N* hodinových taktů, což je velkou nevýhodou této sčítačky. Proto se běžně nepoužívá.



Obr. 44: Paralelní sčítačka se sériovým přenosem

Principiálně nejjednodušší implementace sčítačky je *paralelní sčítačka se sériovým přenosem*, označovaná tzv. *ripple carry adder* (obr. 44). *N*-bitová sčítačka se skládá z *N* jednobitových úplných sčítaček. Nevýhodou sčítačky je velké zpoždění při vytváření sériového přenosu mezi jednotlivými jejími řády. Tato velká kombinační kritická cesta může nepříznivě ovlivnit maximální pracovní frekvenci celého obvodu a je přímo úměrná šířce datového slova *N*. Toto zapojení je možné použít pro sčítání čísel v přímém i doplňkovém kódu a v jazyce VHDL je její popis triviální:

```
c <= a + b;
```

V příloze práce je uveden příklad popisující sčítání dvou *N*-bitových čísel se znaménkem. Zkušební implementace při *N*-bitových operandech (datový typ signed) obsadila v FPGA řady Cyclone III obecně *N* jejich LUTů. V parametrech návrhového systému lze volit parametry syntézy zejména s ohledem na kritickou délku sériového přenosu. Například u systému Quartus je implicitně nastavena maximální dovolená délka sériového přenosu (carry sčítaček) na hodnotu 70 (parametr „Carry Chain Length“) - potom již dochází k automatickému „roztržení“ do separátních řetězců.

Pro vytváření sériového přenosu jsou logické buňky FPGA obvodů architektonicky přizpůsobeny – obsahují speciální kanál pro akceleraci přenosu (carry chain – viz kap. 2.1.1). Nástroje pro syntézu sami zajistí použití příslušných struktur pro zrychlení přenosu. Carry

logika je rychlejší než obyčejná logika, ale mohou být problémy se vzájemným propojením, zvláště pokud je FPGA dosti zaplněný. V každém případě ale zmiňovaný sériový přenos výrazně snižuje maximální pracovní frekvenci – široké čítače jsou schopné běžet na rychlostech řádově desítky MHz. U vyšších rychlostí je lepší řešit problém vylepšením obvodové struktury speciální architekturou. V literatuře lze nalézt velké množství různých zapojení, která především řeší „zrychlování“ přenosů – např. sčítačky s výhybkami (carry-skip adder), sčítačky s predikcí přenosů (carry look-ahead adder), sčítačky s podmíněnými součty (conditional sum adder), asynchronní sčítačky (asynchronous adder) aj. [63], [64], [65], [66]. Principielně nejjednodušším řešením se zdá být použití sériově-paralelní sčítačky, kde dlouhou kaskádu sčítačky zredukujeme na skupinu s 8, 16 nebo 32 bity (kterou řešíme paralelně) a zmiňovanou skupinu potom doplníme registrem a kompletní sčítačku zrealizujeme na principu sériové sčítačky sekvenčním způsobem.

Dalším problémem sčítání může být přetečení – při sčítání dvou *N*-bitových čísel získáme obecně (*N*+1)-bitový výsledek. Je-li výstup sčítačky pouze *N*-bitový, je třeba detekovat přetečení a případně provést následnou korekci výsledku. Obecně mohou podle typů operandů nastat dva případy [67]:

1) Sčítáme-li pouze kladná *N*-bitová čísla v přímém kódu, je přetečení indikováno přenosem z nejvyššího bitu sčítačky.
2) Pracujeme-li s čísly v doplňkovém kódu, přetečení může nastat jen tehdy, pokud sčítáme čísla se stejným znaménkem. Přetečení pak rozpoznáme tak, že výsledek je opačného znaménka než oba operandy. Jinou možností detekce je porovnat přenosy před posledním a za posledním řádem. Jsou-li oba shodné, k přetečení nedošlo.

Jestliže chceme při sčítání dvou *N*-bitových čísel získat (*N*+1)-bitové číslo, je třeba před sčítáním provést znaménkové rozšíření (sign extension) obou sčítanců na *N*+1 bitů. To lze např. ve VHDL provést jednoduše pomocí slučovacího (concatenation) operátoru:

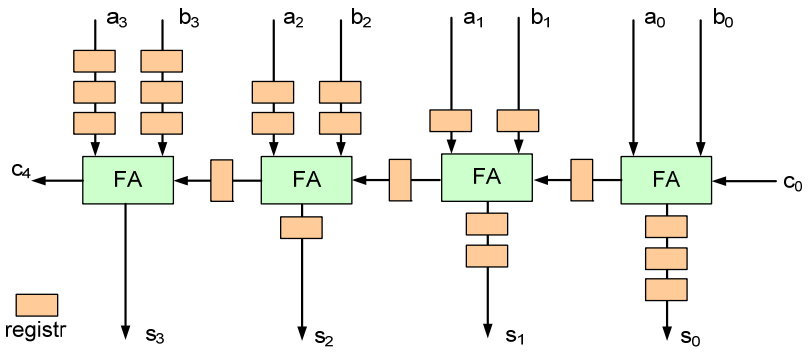
```
c <= (  '0'  & a) + (  '0'  & b);      pro kladná čísla
c <= (a(N-1) & a) + (b(N-1) & b);      pro čísla v doplňkovém kódu
```

Operaci odečítání lze řešit v uvedených přiřazeních buď operátorem „mínus“, případně doplnit sčítačku obvodem pro inverzi znaménka druhého operandu.

V praxi nejčastěji sčítáme proměnný operand s konstantou. Nástroje pro syntézu toto automaticky rozpoznají a sčítačku optimalizují. Implementované zapojení pak nebude úplná sčítačka, ale specializovaný obvod pro přičtení konstanty, např. jedničky (mluvíme pak raději o čítačích).

Významné zvýšení propustnosti při sčítání lze dosáhnout *proudovými sčítačkami* (pipelined adder). Existuje opět mnoho různých variant, na obr. 45 je principiálně nejjednodušší řešení vycházející z paralelní sčítačky se sériovým přenosem [65]. Přenosy z každého řádu jsou odděleny registrem a rovněž vstupy a výstupy jednotlivých řádů jsou

vhodně zpožďovány registry. Výsledek sčítání získáme v podstatě za N hodinových taktů, ale s každým dalším taktem je na výstupech sčítačky další součet. Jednobitovou sčítačku (na obr. 45 blok FA - Full Adder) je možné obecně nahradit i vícebitovou sčítačkou.



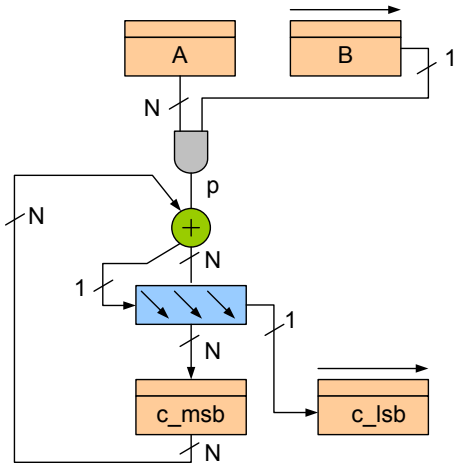
Obr. 45: Sčítačka s proudovým zpracováním

5.2.3 Násobení

Násobení je po sčítání v číslicovém zpracování signálů jedna z nejdůležitějších operací. Existuje opět množství různých hardwarových řešení, jejichž krajními případy je plně sériové a plně paralelní řešení.

Speciálním a nejjednodušším případem násobení je násobení celočíselnou mocninou dvou (obecně 2^r). V tomto případě je násobení převedeno na bitový posuv o r bitů vlevo (zprava doplníme nuly). Předpokládejme dále násobení dvou celých N-bitových čísel a a b. Výsledný součin c může být široký až $2N$ bitů. Nejjednodušším řešení je postupné sčítání (provedeme b-násobně krát součet operandu a). Toto řešení se však pro extrémní pomalost nepoužívá.

Klasická *sériová násobička* zabírá v FPGA obvodu minimální počet buněk, ale je relativně pomalá (vyžaduje N hodinových taktů) a vyžaduje složitější řízení. Násobení



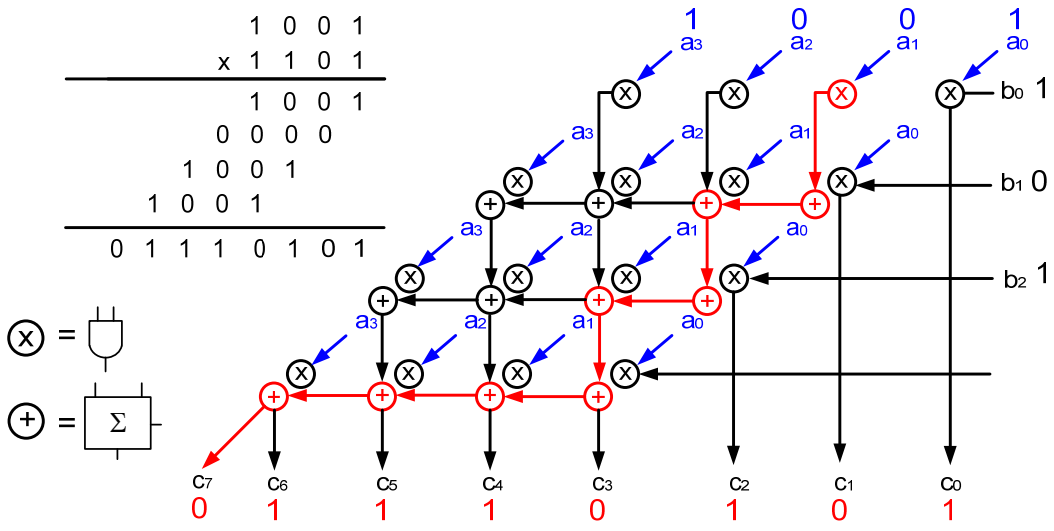
Obr. 46: Zjednodušené schéma sériové násobičky

probíhá podle známého písemného algoritmu (pro nezáporná čísla) a vede na sekvenční obvod – viz obr. 46 [67]. Činitele *a* a *b* jsou uloženy v registrech A a B. Obsah posuvného registru B se s každým hodinovým taktem posouvá vpravo a jeho nejnižší bit je použit pro výpočet dílčího součinu *p*. *N*-bitová sčítačka počítá mezisoučty a nejnižší bit se s každým hodinovým taktem posouvá do posuvného registru *c_lsb*. Po *N* krocích je v registrech *c_msb* a *c_lsb* k dispozici výsledek. Pokud jsou operandy *a* a *b* se znaménkem, musíme v posledním kroku dílčí součin *p* odečíst, protože nejvyšší bit ve dvojkovém doplňku má váhu záporného znaménka. Popisované řešení bylo populární především v menších CPLD obvodech, ale nyní jsou preferovány paralelní algoritmy násobení.

Výrazné zrychlení násobení představuje *paralelní násobička*. Je to v podstatě rozsáhlý kombinační obvod, který vznikne z množství polosčítaček a logických součinů a který opět vychází z algoritmu písemného násobení (obr. 47) [67]. Tato násobička v podstatě vznikne při primitivním zápise v jazyce VHDL přiřazením:

```
c <= a * b;
```

Výsledná implementace násobičky závisí na architektuře použitého FPGA obvodu. Obsahuje-li struktura FPGA hardwarové násobičky (viz kap. 2.2.2), dává návrhový systém většinou přednost jejich použití před aplikací běžné logiky. Při testování v návrhovém systému Quartus firmy Altera bylo implicitně preferováno použití logických elementů (před aplikací DSP bloků) jen při jednodušších násobičkách (pro $N < 5$), a to jak pro čísla se znaménkem, tak pro čísla bez znaménka. V návrhových systémech lze používání DSP bloků při syntéze ovlivňovat volitelnými parametry (např. v systému Quartus parametrem „DSP Block Balancing“). V příloze práce je uveden příklad možného násobení dvou *N*-bitových čísel se znaménkem. Při implementaci 8bitové násobičky s datovým typem signed byl v FPGA řady Cyclone III obsazen pouze jeden DSP blok (resp. násobička), při 32bitové verzi



Obr. 47: Paralelní násobička s příkladem násobení

bylo zaplněno 8 násobiček a 92 LUTů. Pro srovnání stejná 32bitová násobička pouze s datovým typem unsigned zabrala ve stejném FPGA jen nepatrně méně prostoru (8 násobiček a 79 LUTů).

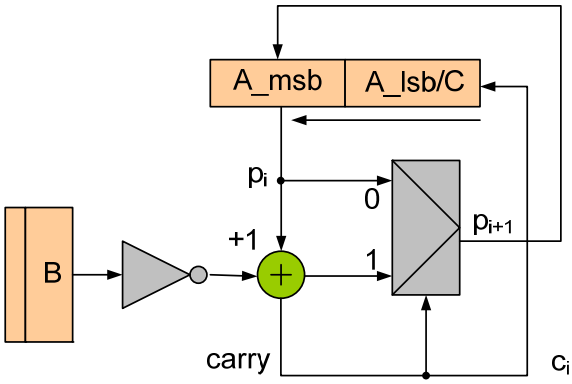
Při použití paralelní násobičky (realizované jak LUTy, tak DSP bloky) se neubráníme vzniku dlouhých kritických cest, které významně snižují maximální hodinovou frekvenci výsledného obvodu. Proto používáme (zvláště pro široká slova) různé sériově paralelní varianty nebo *násobičky s proudovým zpracováním* (pipelined multiplier) [68].

5.2.4 Dělení

Dělení je relativně pomalou aritmetickou operací, a proto se snažíme jeho realizaci nahradit alternativními implementacemi. Nejjednodušším případem je dělení obecnou známou konstantou – v tomto případě provedeme násobení její převrácenou hodnotou.

Dalším jednoduchým případem je, podobně jako u násobení, dělení celočíselnou mocninou dvou (2^r). Realizujeme jej opět bitovým posuvem, tentokrát o r bitů vpravo. Zleva nasouváme nuly (uvažujeme-li čísla bez znaménka) nebo kopii znaménka (u čísel se znaménkem) – mluvíme o znaménkovém rozšíření (sign extension) [67]. Pokud máme čísla v doplňkovém kódu, je třeba si uvědomit, že výsledek je vždy „zaokrouhlován“ směrem k mínus nekonečnu (u záporných čísel je vypočtený podíl větší nebo roven skutečné hodnotě). Extrémní případ z hlediska relativní chyby nastává při dělení čísla $(-1)_D$ libovolnou nezápornou mocninou dvou. Výsledkem je stále $(-1)_D$, tj. $(11111111)_B$. Pokud je zmiňované zaokrouhlování nevhodné, lze nejprve vypočítat absolutní hodnotu čísla, pak jej vydělit a nakonec obnovit původní znaménko.

Předpokládejme nyní obecný dělenec a, dělitel b a výsledný podíl c. Nejjednodušší řešení dělení je postupné odečítání, dokud nedosáhneme čísla menšího nebo rovného nule. Výsledný podíl je pak dán počtem odečtení. Tento algoritmus se však v praxi pro svou pomalost nepoužívá.



Obr. 48: Zjednodušené schéma sériové děličky s nezápornými operandy

Používaným řešením operace dělení je sériová dělička, která pracuje podle známého algoritmu písemného dělení. Jedno z možných řešení implementace děličky pro nezáporné operandy ukazuje obr. 48 [67]. Algoritmus je sekvenční a v každém kroku vypočte novou číslici podílu c_i. Podmínkou správné funkce je, aby byl dělenec i dělitel před dělením normalizován (aby bylo v prvním kroku c_i maximálně 1). To nejjednodušeji zrealizujeme násobením a i b vhodnou mocninou dvou (vhodným posunem). Ve skutečnosti ale budeme dělit $(a.2^n)/(b.2^m)$, výsledek bude tedy třeba vynásobit 2^{m-n} (posunout o m-n bitů). V praxi se používají i další vylepšené varianty děliček popsané např. v [64], [65].

Jinou používanou možností implementace dělení je převod na násobení – operaci $c = a/b$ lze zrealizovat jako $c = a.(1/b)$. Bohužel vytvořit převrácenou hodnotu dělitele b není úplně snadné a většinou ji vytvoříme iteračním algoritmem. Aby byl počet iterací co nejnižší, volíme výchozí hodnotu z převodní tabulky hodnot uložené v paměti [64].

5.3 Implementace základních aritmetických operátorů v pohyblivé řádové čárce

Podstatnou nevýhodou výpočtů v pevné řádové čárce je značně omezený rozsah čísel, které lze do řádové mřížky zapsat. Někdy se tato situace řeší zavedením měřítka S_M, kterým se násobí zobrazované číslo [64]. V číslicové technice je výhodné používat měřítko ve formě celočíselné mocniny dvou, tj. $S_M = 2^r$. Násobení měřítkem je v podstatě ekvivalentní posunu o r bitů. Dokonce každý operand může mít jiné měřítko. Popsané řešení lze použít pouze v případě, že se operandy řádově nemění. Pokud zavedeme proměnnou velikost r, vlastně tento případ zobecníme a dostáváme se k výpočtům v pohyblivé řádové čárce.

Naznačme si postupy řešení základních výpočtů s čísly ve formátu pohyblivé řádové čárky (ve dvojkové soustavě). Předpokládejme operace se dvěma operandy $a = M_{N1}.2^{E1}$ a $b = M_{N2}.2^{E2}$. Pokud uvažujeme o formátech IEEE, je třeba na počátku algoritmu zkontrolovat, zda operandy nemají „speciální“ hodnoty (viz kap. 5.1). Pokud ano, lze většinou výsledek snadno určit - záleží na konkrétních hodnotách (±0, ±∞, NaN).

Naznačme si nejprve postup při **sčítání**, resp. **odečítání** uvažovaných dvou operandů a a b:

1) Zjistí se, který z exponentů je větší, a vypočítá se rozdíl mezi exponenty.
2) Když je rozdíl nulový, tento bod se přeskočí. Jestliže je $E1 > E2$, posouvá se mantisa M_{N2} o tolik bitů doprava, o kolik činí rozdíl exponentů; jestliže je $E2 > E1$, posouvá se doprava mantisa M_{N1}. Výhodou exponentů ve formátu IEEE 754 je, že jsou vždy kladné, a nemusí se testovat jejich znaménka.
3) Podle znamének obou hodnot se mantisy sčítají nebo odečítají.
4) Provede se normalizace výsledku. Jestliže mantisa výsledku přetekla (overflow), provede se posun o jeden bit doprava a exponent se zvýší o jedničku. Pokud mantisa nemá nejvyšší bit jedničkový, posouvají se bity doleva. O počet posunů se sníží hodnota exponentu výsledku. Následně se mantisa zaokrouhlí.

5) Zkontroluje se výsledek exponentu. Je-li větší, než může být maximální hodnota exponentu, výsledek přetekl. Výsledkem je $+\infty$, resp. $-\infty$ (podle výsledného znaménka). Naopak, když je výsledný exponent menší než minimální hodnota exponentu, výsledek podtekl (underflow) – výsledkem je ± 0 (podle znaménka).

Postup při **násobení** dvou hodnot ve formátu FP je následující:

1) Vypočte se znaménko výsledku (ze znaménkových bitů operandů využitím hradla XOR).
2) Vypočte se výsledná mantisa vynásobením obou mantis operandů. Pokud výsledná mantisa přeteče, je bitově posunuta doprava o potřebný počet bitů a exponent je o tuto hodnotu zvýšen.
3) Vypočte se výsledný exponent sečtením obou exponentů jednotlivých operandů. Od tohoto výsledku se musí u FP čísel ve formátu IEEE odečíst posun aditivního kódování K (jinak by byl posun $2K$, protože u každého exponentu již je K přičteno).
4) Provede se normalizace výsledku. Provedou se posuny mantisy vpravo, resp. vlevo a úměrně počtu posunů se zvýší, resp. sníží exponent. Následně se mantisa zaokrouhlí.
5) Zkontroluje se výsledek exponentu, obdobně jako v bodu 5 při sčítání.

Algoritmus pro **dělení** bude obdobný jako součin, pouze mantisy budeme dělit a exponenty odečítat. U exponentů v aditivním kódu zde musíme naopak konstantu K k výslednému exponentu přičíst.

Z naznačených postupů FP aritmetiky je patrné, že výsledné hardwarové řešení bude jistě složitější, a tím také pomalejší, ve srovnání s FX aritmetikou. Často jsou FP operace řešeny proudovým zpracováním a k získání výsledku je třeba několik hodinových taktů. Operace s mantisou jsou částečně podobné operacím s pevnou řádovou čárkou, ale v FX aritmetice se častěji používá reprezentace čísel ve dvojkovém kódu. O operacích v pohyblivé řádové čárce je možné se více dočíst např. v [64] a [65].

Návrh vlastních obvodových řešení ve FP aritmetice není jednoduchý, a rozhodně není určen pro málo zkušené návrháře. Pokud se ukáže potřeba realizace vybraných FP operací, lze bez problémů použít vhodné konfigurovatelné makrobloky, které bývají nyní již běžnou součástí návrhových systémů. Například u systému Quartus II (verze 9.0) jsou pro FP operace k dispozici následující megafunkce [70]: sčítání/odečítání, násobení, dělení, druhá odmocnina, exponenciální funkce, inverzní funkce, přirozený logaritmus, absolutní hodnota, komparátor, inverze matice, inverze matice, násobení matic, konvertor mezi různými formáty FX a FP čísel aj. Tyto makrobloky využívají ke své implementaci jak běžné logické elementy, tak někdy i speciální struktury (DSP bloky, blokovou RAM), a mohou být tedy částečně závislé na architektuře konkrétní řady FPGA obvodů.

Závěr

Práce se zaměřuje na oblast číslicových systémů realizovaných na bázi programovatelných hradlových polí. Využívá dosavadní více jak 20leté pedagogické a praktické zkušenosti autora s návrhem těchto obvodů, člení a zobecňuje nejen vlastní znalosti, ale zpracovává velké množství poznatků z naší i zahraniční literatury. Třídí a vyjasňuje odbornou terminologii používanou v literatuře mnohdy nejednotně a snaží se o hledání případných vhodných českých ekvivalentních názvů.

Významná část práce se věnuje architektuře FPGA obvodů, jejím základním i speciálním strukturním prvkům, jejich přednostem či omezením. Všímá si rozdílů v jednotlivých konkurenčních architekturách zejména dvou nejvýznamnějších výrobců – firem Xilinx a Altera. Snaží se ukázat na využitelnost jednotlivých prvků, naznačuje vzájemné souvislosti a případné trendy dalšího vývoje. Jsou zde rovněž zmiňovány principy programování FPGA obvodů, jejich výrobní technologie a možnost rekonfigurace.

Část práce věnovaná metodice návrhu programovatelných obvodů je spíše přehledová a slouží k ujasnění si postupů a návazností v průběhu návrhu. Může sice v některých konkrétních částech naznačit návrháři alternativy postupu, ale hlavně je tato kapitola psána jako jakýsi úvod ke stěžejní kapitole věnované systémovému návrhu.

Systémový návrh ovlivňuje výslednou velikost, dynamické parametry, energetickou spotřebu a zejména spolehlivost navržených obvodů. V práci je diskutován postup tohoto návrhu, jeho specifikace na vyšších úrovních abstrakce a využívání různých forem makrobloků. Důležitou částí jsou zásady návrhu synchronních systémů včetně rozboru a odstraňování metastabilit, které jsou stále aktuálnějším problémem současných návrhů s mnoha časovými doménami. Respektováním naznačených zásad lze dosáhnout dlouhodobě spolehlivě fungujících řešení. Většinu z uvedených pravidel je možné zobecnit a použít i pro oblast návrhu ASIC obvodů s ještě vyšší hustotou integrace.

Zajímavá je oblast návrhu aritmetických operací v FPGA obvodech, která je v podstatě konkretizací vybraných úvah systémového návrhu. Tuto část jistě ocení nejen návrháři systémů pro zpracování dat, ale třeba i návrháři různých řídicích algoritmů. Je zde stručně a přehledně zpracována problematika kódování binárních čísel jak v pevné, tak v pohyblivé řádové čárce. Dále jsou ukázány výhody a nevýhody realizace vybraných variant základních algoritmů aritmetických operací implementovaných v FPGA obvodech. Na jejich řešení je dobře patrná mnohoznačnost syntézy z hlediska různých kritérií.

Vybrané části práce jsou doplněny většinou jednoduchými názornými příklady, vesměs psanými v jazyce VHDL a kompilovanými v návrhovém systému Quartus II verze 9.0. (full version). Jako cílový FPGA obvod byl volen EP3C5F256A7 z řady Altera Cyclone III. Popis

jak VHDL jazyka, tak konkrétní práce s návrhovým systémem je mimo rámec této práce. Příklady slouží zejména pro ověření realizovatelnosti, porovnání výsledných parametrů návrhu a ke zjišťování vlivu nastavení volitelných parametrů nástrojů syntézy. Zmiňované příklady jsou uvedeny v elektronické příloze práce na CD médiu.

Věříme, že text práce bude přínosem jak pro méně zkušené hardwarové návrháře FPGA obvodů, tak pro studenty různých stupňů studia nejen naší fakulty.

Literatura

[1] Kolouch, J.: Programovatelné logické obvody a hradlová pole – moderní stavební prvky číslicových systémů. Automatizace, č. 1, 2009, str. 46-49.

[2] Xilinx: Virtex-5 EasyPath FPGAs. [online], [cit. 2009-12-21], URL: <http://www.xilinx.com/publications/prod_mktg/pn2019.pdf>

[3] Altera: About HardCopy ASIC series. [online], [cit. 2009-12-21], URL: <http://www.altera.com/products/devices/hardcopy-asics/about/hrd-index.html>

[4] Xilinx: The Company Profile … Xilinx. [online], [cit. 2009-12-21], URL: <http://www.xilinx.com/esp/automotive/xilinxcompanyprofile_Hansen_Report.pdf>

[5] Altera: Step 1: Defining Stratix II Logic Structure. [online], [cit. 2009-12-21], URL: <http://www.altera.com/products/devices/performance/high_performance/define/per-define.html>

[6] Altera: Stratix III Device Family Architecture. [online], [cit. 2009-12-21], URL: <http://www.altera.com/products/devices/stratix-fpgas/stratix-iii/overview/architecture/performance/st3-alm-structure.html>

[7] Advantages of the Virtex-5 FPGA 6-Input LUT Architecture. [online], [cit. 2009-12-21], URL: <http://www.xilinx.com/support/documentation/white_papers/wp284.pdf>

[8] Xilinx: Spartan-6 FPGA Configurable Logic Block. [online], [cit.2009-12-21], URL: <http://www.xilinx.com/support/documentation/user_guides/ug384.pdf>

[9] Xilinx: Spartan and Spartan-XL Families Field Programmable Gate Arrays. [online], [cit. 2009-12-21], URL: <http://lhcb-online.web.cern.ch/lhcb-online/ecs/ccpc/docs/XCS05xl%20datasheet.pdf>

[10] Daněk, M.: Programovatelná hradlová pole – FPGA. Automa, č.2, 2006, s. 9-13

[11] Xilinx: Virtex-6 Family Overview. [online], [cit. 2009-12-21], URL: <http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf>

[12] Altera: Stratix IV Device Handbook. [online], [cit. 2009-12-21], URL: <http://www.altera.com/literature/hb/stratix-iv/stratix4_handbook.pdf>

[13] Xilinx: Spartan-6 Family Overview. [online], [cit. 2009-12-21], URL: <http://www.xilinx.com/support/documentation/data_sheets/ds160.pdf>

[14] Xilinx: Spartan-3 FPGA Family Data Sheet. [online], [cit. 2009-12-21], URL: <http://www.xilinx.com/support/documentation/data_sheets/ds099.pdf>

[15] Kolouch, J.: Možnosti realizace pamětí RAM v obvodech FPGA. Elektrorevue, 2003, č. 27, [online], [cit. 2009-12-21], URL: <http://www.elektrorevue.cz/clanky/03027/index.html>

[16] Kolouch, J.: Implementace procesorů v obvodech FPGA. Elektrorevue, 2008, ISSN 1213-1539, [online], [cit. 2009-12-21], URL: <http://www.elektrorevue.cz/cz/clanky/power-electronics-1/0/implementace-procesoru-v-obvodech-fpga/>

[17] Altera: Implementing High Performance DSP Functions in Stratix & Stratix GX Devices. [online], [cit. 2009-12-21], URL: <http://www.altera.com/literature/hb/stx/ch_7_vol_2.pdf>

[18] Altera: Clock Networks and PLLs in Cyclone III Devices. [online], [cit. 2009-12-21], URL: <http://www.altera.com/literature/hb/cyc3/cyc3_ciii51006.pdf>

[19] Xilinx: Embedded Processor Block in Virtex-5 FPGAs. Reference Guide. [online], [cit. 2009-12-21], URL: <http://www.xilinx.com/support/documentation/user_guides/ ug200.pdf>

[20] Krehbiel, J.: Powering FPGA-based Boards. FPGA and Structured ASIC Journal, [online], [cit. 2009-12-21], URL: <http://www.fpgajournal.com/articles/20040713_intersil.htm>

[21] Actel: High-Volume nano FPGAs. [online], [cit. 2009-12-21], URL: <http://www.actel.com/ documents/nano_Technology_WP.pdf>

[22] Shang, L. – Kaviani, A. – Bathala, K.: Dynamic Power Consumption in Virtex-II FPGA Family. [online], [cit. 2009-12-21], URL: <http://ecee.colorado.edu/~lshang/papers/shang02feb.pdf>

[23] Hansen, L. –Thomas, T.: Complete FPGA and CPLD Power Analysis. [online], [cit. 2009-12-21], URL: <http://www.xilinx.com/publications/xcellonline/xcell_53/xc_pdf/xc_power53.pdf>

[24] Xilinx: Power Consumption in 65 nm. [online], [cit. 2009-12-21], URL: <http://www.xilinx.com/ support/documentation/white_papers/wp246.pdf>

[25] National Semiconductor: Power Management Design Guide for Altera FPGAs and CPLDs. [online], [cit. 2009-12-21], URL: <http://www.national.com/appinfo/power/files/NationalAlteraDesignGuide.pdf>

[26] Texas Instruments: Tips for successful power-up of today's high-performance FPGAs. [online], [cit. 2009-12-21], URL: <http://focus.ti.com/lit/an/slyt079/slyt079.pdf>

[27] Altera: Configuration Elements. [online], [cit. 2009-12-21], URL: <http://www.altera.com/literature/ds/dsconfig.pdf>

[28] Actel: Technology Solutions - Power. [online], [cit. 2009-12-21], URL: <http://www.actel.com/ products/solutions/power/default.aspx>

[29] Salcic, Z. – Smailagic, A.: Digital Systems Design and Prototyping. Kluwer, 2000, second edition, ISBN: 0-7923-7920-9.

[30] Lattice: LatticeXP2 Family Handbook. [online], [cit. 2009-12-21], URL: <http://www.latticesemi.com/documents/HB1004.pdf>

[31] Altera: Design Security in Stratix III Devices. [online], [cit. 2009-12-21], URL: <http://www.altera.com/literature/hb/stx3/stx3_siii51014.pdf>

[32] Maxim: Xilinx FPGA IFF Copy Protection with 1-Wire SHA-1 Secure Memories. [online], [cit. 2009-12-21], URL: <http://www.maxim-ic.com/appnotes.cfm/an_pk/3826>

[33] Altera: An FPGA Design Security Solution Using a Secure Memory Device. [online], [cit. 2009-12-21], URL: <http://www.altera.com/literature/wp/wp-01033.pdf >

[34] Altera: Device Package Information Data Sheet. [online], [cit. 2009-12-21], URL: <http://www.altera.com/literature/ds/dspkg.pdf>

[35] Šťastný J.: Programovatelná hradlová pole. Automatizace, 1/2008, s. 9-14.

[36] Kloub, J.: Architektura systému pro nekonfigurovatelný komunikační terminál. In: Počítačové architektury a diagnostika 2008, Liberec 15.-17. Září 2008.

[37] Sekanina, L.: Evolvavle Components: From Theory to Hardware Implementation. Springer, 2004, ISBN: 3540403779

[38] Lattice Semiconductor: Minimizing System Interruption During Configuration Using TransFR Technology. [online], [cit. 2009-12-21], URL: <http://www.latticesemi.com/lit/docs/technotes/tn1087.pdf>

[39] Daněk, M. – Honzík, P. – Kadlec, J. – Matoušek, R. – Pohl, Z.: Platforma s částečnou dynamickou rekonfigurací FPGA. Automa, č.5, 2006, s. 40-43

[40] Schmidt, J.: Architektura programovatelných logických obvodů. Sdělovací technika, 2/2000, s. 16-19.

[41] Sherwani, N.: Algorithms for VLSI Physical Design Automation. Kluwer, 2002, third edition, ISBN: 0-7923-8393-1.

[42] Šťastný J.: Návrh obvodů založených na programovatelných hradlových polích. Automatizace, 5/2008, s. 317-321.

[43] Keating, M. – Bricaud, P.: Reuse Methodology Manual. Springer, 2002, ISBN: 1402071418.

[44] Laipert, M. – Kolář, M. – Horčík, Z: Systémový návrh zakázkových integrovaných obvodů. Skriptum ČVUT FEL, Praha 1992

[45] Altera: Nios II Performance Benchmarks. [online], [cit. 2009-12-21], URL: <http://www.altera.com/literature/ds/ds_nios2_perf.pdf>

[46] Dřínovský, M. - Šťastný, J.: Implementace sériových aritmetických operátorů na moderních programovatelných hradlových polích. Elektrorevue, 2009, č. 32, [online], [cit. 2009-12-21], URL: <http://www.elektrorevue.cz/cz/clanky/power-electronics-1/0/implementace-seriovych-aritmetickych-operatoru-na-modernich-programovatelnych-hradlovych-polich/>

[47] Clive "Max" Maxfield: The Design Warrior's Guide to FPGAs: Devices, Tools and Flows. Newnes, 2004, ISBN: 0-7506-7604-3

[48] Netolická, K.: Equivalence Checking of Retimed Circuits. [online], [cit. 2009-12-21], URL: <http://dspace.mit.edu/bitstream/handle/1721.1/32104/62324849.pdf>

[49] Pinker, J. – Poupa, M.: Číslicové systémy a jazyk VHDL. BEN – technická literatura, Praha 2006, ISBN: 80-7300-198-5.

[50] Altera: System Quartus II, version 9.0, help: State Machine Processing logic option.

[51] Xilinx: System ISE version 9.1, help: Language Templates – VHDL – Synthesis Constructs – Coding Examples – State Machines – Info: Finite State-machines.

[52] Shen, J. P. – Lipasti, M. H.: Modern processor design – fundamentals of superscalar processors. McGraw-Hill Science, 2004, ISBN: 0-07-057064-7

[53] Altera: Stratix Series FPGA Low Power Consumption Features. [online], [cit. 2009-12-21], URL: <http://www.altera.com/products/devices/stratix-fpgas/about/low-power-consumption/stx-power-about.html>

[54] Šťastný J.: Návrh specifických struktur na programovatelných hradlových polích. Automatizace, 11/2008, s. 688-692

[55] Altera: Understanding Metastability in FPGAs. [online], [cit. 2009-12-21], URL: <http://www.altera.com/literature/wp/wp-01082-quartus-ii-metastability.pdf>

[56] Altera: Metastability in Altera Devices. [online], [cit. 2009-12-21], URL: <ftp://ftp.altera.com/pub/lit_req/document/an/an042.pdf>

[57] EDACafe: ASICs, Chapter 6.4 – AC Input. [online], [cit. 2009-12-21], URL: <http://www10.edacafe.com/book/ASIC/CH06/CH06.4.php#pgfId=19221>

[58] Actel: Metastability Characterization Report for Actel Flash FPGAs. [online], [cit. 2009-12-21], URL: <http://www.actel.com/documents/Flash_Metastability_HBs.pdf>

[59] Verma, S., Dabare, A. S.: Understanding Clock Domain Crossing Issues. [online], [cit. 2009-12-21], URL: <http://www.edadesignline.com/howto/205201913>

[60] Adventures in ASIC Digital Design: The Double Edge Flip Flop. [online], [cit. 2009-12-21], URL: <http://asicdigitaldesign.wordpress.com/2007/07/31/the-double-edge-flip-flop/>

[61] Adventures in ASIC Digital Design: Dual Edge Binary Counters + Puzzle. [online], [cit. 2009-12-21], URL: <http://asicdigitaldesign.wordpress.com/2009/06/24/dual-edge-binary-counters/>

[62] Altera: Using Tri-State Buses for Bidirectional Communication. [online], [cit. 2009-12-21], URL: <http://www.altera.com/support/examples/ged/tri_state.html>

[63] Mlynek, D. – Leblebici, Y.: Design of VLSI Systems, Chapter 6 – Arithmetic for digital systems. [online], [cit. 2009-12-21], URL: <http://lsiwww.epfl.ch/LSI2001/teaching/webcourse/ch06/ch06.html>

[64] Pluháček, A.: Projektování logiky počítačů. [Skripta], Vydavatelství ČVUT, Praha 2003, ISBN: 80-01-02145-9

[65] Ercegovac, M. D. – Lang, T.: Digital Arithmetic. Elsevier Science, San Francisco, USA, 2004, ISBN: 1-55860-798-6

[66] Marcinčák, M.: Přenosy ve sčítačkách. [Bakalářská práce], ČVUT FEL, Praha 2008

[67] Šťastný J.: Návrh aritmetických operátorů na FPGA. Automatizace, 2/2009, s. 71-74

[68] Lewis J.: Coding a 40x40 Pipelined Multiplier in VHDL. [online], [cit. 2009-12-21], URL: <http://www.synthworks.com/papers/VHDL_RTL_Pipelined_Multiplier_MAPLD_2002_S_BW.pdf>

[69] Kaňka, M.: Implementace aritmetiky čísel s pohyblivou řádovou čárkou v obvodech FPGA. [Bakalářská práce], TUL FM, Liberec 2008

[70] Altera: Floating-Point Megafunctions User Guide. [online], [cit. 2009-12-21], URL: <http://www.altera.com/literature/ug/ug_altfp_mfug.pdf>

Seznam příloh v elektronické formě na CD

habilitace-kolar.pdf - habilitační práce ve formátu pdf

Příklady v jazyce VHDL

ram_dual.vhd - dvouportová paměť RAM realizovaná procesy
ram_dual_ip.vhd - dvouportová paměť RAM realizovaná IP blokem
share.vhd - ukázka sdílení prostředků
mealy.vhd - stavový automat typu Mealy
moore.vhd - stavový automat typu Moore
dual_dff.vhd - dvouhranový klopný obvod
dual_counter.vhd - dvouhranový čítač
three_state_bus.vhd - třístavová jednosměrná sběrnice
bidir_bus.vhd - třístavová obousměrná sběrnice
inversion.vhd - operace inverze znaménka v pevné řádové čárce
adder.vhd - operace sčítání v pevné řádové čárce
multiplier.vhd - operace násobení v pevné řádové čárce